Conception d'architectures reconfigurables dynamiquement : Du silicium au système

Sébastien Pillement

N° d'ordre : xxx

Habilitation à Diriger les Recherches
présentée
DEVANT L'UNIVERSITÉ DE RENNES 1

par
Sébastien Pillement

Équipe d'accueil :	Institut de Recherche en Informatique et Systèmes Aléatoires - CAIRN
École Doctorale :	Mathématiques, Télécommunications, Informatique, Signal, Systèmes et Électronique
Composante Universitaire :	Institut Universitaire de Technologie de Lannion

---

Conception d'architectures reconfigurables dynamiquement : Du silicium au système

---

soutenue publiquement le 22 octobre 2010
devant la commission d'examen composée de :

Président du jury :	
Daniel Etiemble	Professeur des Universités, Université de Paris Sud
Rapporteurs :	
Christophe Bobda	Professeur des Universités, Université de Postdam
Michel Paindavoine	Professeur des Universités, Université de Bourgogne
Lionel Torres	Professeur des Universités, Université de Montpellier II
Examinateurs :	
Dominique Lavenier	Directeur de Recherche, CNRS
Didier Demigny	Professeur des Universités, Université de Rennes 1

A ma famille,

qui est et restera le moteur de ma vie.

Je vous aime

Table des matières

1 Contexte général

A l'heure ou l'Homme s'apprête à mesurer la masse du neutrino, il est clair que l'évolution des technologies est liée à notre capacité à appréhender l'infiniment petit. Cette miniaturisation est, depuis toujours, le moteur de l'innovation et de l'intégration des systèmes électroniques. On est passé du poste de travail posé sur le bureau au portatif, du portatif au portable, du portable au pocket. Dans les prochaines années, virtuellement tous les objets de la vie courante auront des capacités de calculs, et cet ensemble supportera le concept d'*Everyware*[1] [36]. Dans ce modèle, des dispositifs de traitement, petits et peu coûteux, connectés en réseau (intégrés et distribués dans les objets de la vie quotidienne), visent à créer un espace quotidien intelligent. Aussi immédiat d'utilisation que l'est aujourd'hui l'électricité, ils permettent d'interagir directement avec les objets de l'environnement domestique ou public. Par exemple, un environnement informatique ubiquitaire domestique pourrait relier tous les contrôles de l'éclairage et de l'environnement avec des capteurs biométriques individuels cousus dans l'habillement pour moduler en conséquence l'éclairage et le chauffage dans une chambre imperceptiblement et sans interruption. Un autre scénario commun pose le principe de réfrigérateurs proposant des menus variés en fonction des aliments qu'ils contiennent et avertissent lorsque des denrées viennent à manquer ou sont avariées. De nouveaux systèmes embarqués ont été développés et permettent d'envisager de nouvelles perspectives comme les vêtements intelligents "i-wear" permettant la détection de la chute ou de surveiller des paramètres médicaux de personnes malades, ou encore les capsules bourrées d'électronique permettant de faire de l'imagerie médicale par l'intérieur du corps.

Le concept d'*Everyware* nécessite d'intégrer des microprocesseurs dans les objets de la vie quoti-

1. Ce terme, inventé par l'auteur, vise à unifier les dénominations de ce paradigme. Des synonymes plus ou moins proches sont l'informatique pervasive (pervasive computing), l'intelligence ambiante [ambient intelligence] où encore les objets enfouis (calcul physique [physical computing], l'Internet des choses [Internet of things], les choses qui pensent [things that think] et le "spime").

dienne, de permettre à ces microprocesseurs de communiquer entre eux et avec l'environnement par des réseaux ad'hoc, et enfin de définir des interfaces permettant aux usagers de contrôler et d'interagir avec ces objets de la manière la plus naturelle possible. La complexité grandissante des services et des réseaux, nécessite alors le développement de systèmes actifs, capables de gérer leurs ressources à travers un environnement distribué et hétérogène. De plus, ces systèmes doivent pouvoir supporter de multiples standards et s'adapter de manière dynamique aux services requis (voix, données haut ou bas débit, etc.). L'adaptabilité des systèmes et les performances de calcul requises poussent à la définition d'architectures reconfigurables et programmables [2]. Ces architectures doivent, en plus, répondrent au critère de faible consommation car elles seront enfouies ou embarquées.

De nouveaux types d'applications qui doivent s'adapter à des conditions d'exécution changeantes avec le temps et l'espace, font leurs apparitions (radio-logicielle, codage multimédia, réalité augmentée, ...). Parallèlement, elles requièrent de plus en plus de puissance de calcul, afin de respecter des contraintes temps réel devenues très exigeantes. Il est donc nécessaire de définir des architectures hétérogènes autorisant un compromis entre de hautes performances de calculs, de la flexibilité et une certaine efficacité en consommation d'énergie. Ceci conduit alors les concepteurs à s'orienter vers des solutions logicielles et matérielles basées sur des unités de calcul travaillant en parallèle. L'utilisation d'architectures flexibles et fortement parallèles implique que le support de communication soit capable, lui aussi, de s'adapter aux nouveaux éléments présents dans le système.

La diminution de la taille des transistors va de paire avec l'augmentation des coûts de fabrication des systèmes. Ainsi, si la fabrication des masques en technologie 65 nm revient à 1,5 million $, il en coûte le double pour une fabrication en technologie 45 nm. Dans le même temps, les coûts de développement en 2010 sont estimés à 60 millions $ répartis entre le développement du logiciel des futurs systèmes (2/3 du coût) et le matériel (tiers restant). Cette contrainte économique forte, pousse alors au développement de systèmes généralistes permettant la réutilisation de l'architecture et des codes. Cette propriété tend à développer des systèmes supportant le plus d'applications possible. D'autre part, les nouveaux systèmes doivent supporter le passage à l'échelle (scalabilité). Ceci afin de rentabiliser à long terme les développements en supportant des applications non définies lors de la conception de l'architecture sous-jacente. Une solution, est alors de concevoir des systèmes sur puce reconfigurable (RSoC - pour Reconfigurable System on Chip) permettant l'adaptation de l'architecture. Une autre approche permettant la réutilisation, est d'abstraire la complexité des architectures en rajoutant une surcouche logicielle de type système d'exploitation (OS – Operating System). Cette idée permet en outre de supporter des systèmes plus globaux et distribués.

Pour supporter le concept d'*Everyware*, les architectures doivent donc intégrer différents types

2. Il est difficile de différentier de manière claire les concepts de reconfiguration et de programmation. Ces deux concepts introduisent de la flexibilité dans l'exécution des applications. Nous ferons la distinction par la capacité des architectures reconfigurables à modifier leurs chemins de données de manière physique (réorganisation des éléments de calcul).

d'opérateurs, êtres flexibles et supporter du parallélisme. Les architectures reconfigurables dynamiquement (ARD) permettent d'effectuer ce saut de performances. Leur concept de base est simple puisqu'il s'agit d'adapter la structure du matériel au type d'algorithme qu'elles ont en charge, et ce à la volé, c'est à dire pendant l'exécution même de l'application. Cette flexibilité permet entre autres de pouvoir s'adapter efficacement à un contexte applicatif de plus en plus dynamique. En contrepartie, l'utilisation de ces architectures pose néanmoins des problèmes en termes de contrôle, de gestion et de conception.

Mes travaux de recherches se placent clairement dans le cadre de la définition et de la gestion de systèmes reconfigurables dynamiquement. Ma préoccupation principale est alors la gestion de la complexité. Dans le domaine de la conception des systèmes, cette préoccupation intervient à différents niveaux :

- Complexité technologique. Nous travaillons à l'heure actuelle aux frontières de la physique. Certains phénomènes non pris en compte jusqu'alors deviennent problématiques. Il faut alors concevoir les circuits en prenant en compte la fiabilité, le rendement de fabrication, ou encore la performance.
- Complexité architecturale. Elle vient du nombre d'objets intégrables dans les systèmes actuels qu'il faut assembler (concevoir) mais qu'il faut aussi gérer (programmer). A ce niveau, l'augmentation des performances, au sens large du terme (traitement, consommation), fait apparaître un problème de gestion du parallélisme.
- Complexité de conception. La programmation ainsi que les méthodologies de conception se doivent d'être adaptées. Cette difficulté, relativement à des contextes applicatifs nouveaux, nécessite une efficacité dans la conception pour obtenir dans des temps raisonnables de nouveaux systèmes performants et flexibles.

Mes activités de recherches sont directement liées aux trois préoccupations mentionnées ci-dessus. Afin de proposer des solutions aux problèmes rencontrés, j'ai développé trois axes de recherche :

Axe 1 : Architecture Actuellement, les gains en performance et en flexibilité requis sont adressés par la définition d'architectures massivement parallèles comme en témoigne le processeur Fermi de la société Nvidia [61] (processeur graphique comportant 512 unités de traitement). Si cette solution est viable dans le domaine de l'informatique haute performance, il en va différemment pour les systèmes embarqués qui ont des contraintes, notamment de coût et de consommation, incompatibles avec ce type d'approches.

Dans cet axe, je travaille à la définition d'architectures reconfigurables efficaces en consommation et performantes. Une architecture se devant d'être exploitée je propose aussi les méthodologies de conceptions associées.

Axe 2 : Interconnexions Les architectures futures intégrerons une multitude de blocs fonctionnels hétérogènes. La communication entre ces différents blocs (processeurs généra-

listes, processeurs spécifiques, mémoires, IP, etc.) constitue alors un véritable défi. Ainsi, le problème n'est plus tant de concevoir des composants efficaces, mais surtout de les assembler efficacement afin d'obtenir une architecture qui respecte les contraintes de performances, de coût et de fiabilité.

Cet axe regroupe mes travaux autour de l'interconnexion dans les systèmes RSoC. Basées sur la notion de réseau sur puce, nous avons proposé des approches pour améliorer la fiabilité et réduire la consommation des liaisons et supporter la flexibilité requise.

Axe 3 : Gestion Un des grands enjeux des sciences et technologies de l'information et de la communication (STIC) est certainement l'exploitation efficace des capacités d'intégration et de calculs disponibles. Le modèle architectural massivement parallèle, hétérogène et distribué pose alors de grands défis pour une gestion efficace de ces architectures.

Mes travaux autour des systèmes d'exploitation pour architectures reconfigurables prennent place dans cet axe. Je travaille sur des services classiques de gestion comme l'ordonnancement ou les communications, mais j'ai aussi des activités autour du support de la tolérance aux fautes.

Ce manuscrit s'articule ainsi autour de cinq chapitres :

- le premier chapitre est une présentation générale de mon parcours. J'y présente, de manière synthétique, l'ensemble de mes activités d'enseignant-chercheur. Ce premier chapitre dresse un bilan de mes implications dans les activités d'encadrements, de valorisations et d'administration de la recherche. Je présente également un résumé de mes perspectives de recherche.
- le chapitre 2 détaille mes activités de recherche autour de la conception d'architectures reconfigurables dynamiquement. Une analyse des contraintes est donnée afin d'expliciter les choix et les solutions retenues.
- dans le chapitre 3, j'adresse la problématique de la conception et de l'intégration d'interconnexions flexibles et efficaces. L'efficacité s'entend ici en terme de consommation et de fiabilité des liaisons physiques entre les composants du système. La flexibilité est supportée par l'intégration de la notion de réseau sur puce.
- le chapitre 4 présente mes activités dans le domaine des méthodologies de gestion des architectures reconfigurables embarquées. Notamment, la problématique de la modélisation, de l'exploration et de la conception de systèmes d'exploitation temps réels embarqués y est présentée. Le deuxième point abordé vise à proposer l'utilisation de la propriété de reconfiguration dynamique en vue de concevoir des systèmes sûr de fonctionnement.
- dans le dernier chapitre je dresse un bilan de mes activités de recherche, et propose des perspectives à long terme de mes travaux. Les perspectives court - moyen terme seront, quant à elles, exposées tout au long du document.

Chapitre

1 Synthèse des travaux

Résumé : Ce chapitre est un aperçu rapide et concis de l'ensemble de mes différentes activités d'enseignant-chercheur. J'y aborde l'ensemble de mes activités, y compris l'enseignement, avant de développer certains de mes axes de recherches dans les chapitres suivants.

Sommaire

A Curriculum Vitæ

Informations personnelles

Sébastien PILLEMENT
38 ans
Marié, 3 enfants
14 rue André Breton
22 300 Lannion
tél : 02 96 48 95 64

Coordonnées professionnelles

IRISA/CAIRN - ENSSAT
6 Rue de Kérampont
Téléphone : 02 96 46 91 66
Télécopie : 02 96 46 66 75
Sébastien.Pillement@irisa.fr
http ://cairn.irisa.fr/

Grades et titres universitaires

1995-1998 : Doctorat en Electronique, Optronique et Système.

Titre :	Méthodologies d'évaluation et de prototypage des systèmes numériques intégrés
Laboratoire :	LIRMM - Université de Montpellier II
Directeur de thèse :	M. Robert, LIRMM, Université de Montpellier II

soutenue le 7 décembre 1998 à Montpellier devant le jury composé de :

Président du jury :	G. Cambon, LIRMM, Université de Montpellier II
Rapporteurs :	S. Weber, LIEN, Université de Nancy I
	M. Paindavoine, LE2I, Université de Bourgogne
Examinateurs :	L. Torres, LIRMM, Université de Montpellier II
	B. Rouzeyre, LIRMM, Université de Montpellier II
	D. Barthel, CNET, Centre France Télécom de Meylan
	J. Carrabina, Université Autonome de Barcelone

1994-1995 : DEA *Conception Assistée des Systèmes Informatiques, Automatiques et Micro-électronique*, option Micro-électronique, Université Montpellier II, mention Assez Bien.

1993-1994 : Maîtrise d'*Electronique, Electrotechnique et Automatique*, Université de Toulon et du Var, mention Assez-Bien.

1992-1993 : Licence d'*Electronique, Electrotechnique et Automatique*, Université de Toulon et du Var, mention Assez-Bien.

1991-1992 : Licence *Ingénierie électrique*, Université de Toulon et du Var, mention Bien.

1989-1991 : DUT *Génie Electrique et Informatique Industrielle*, Institut Universitaire de Technologie de Toulon-la Garde.

Expériences professionnelles

Depuis 1999 :	Maître de Conférences à l'IUT de Lannion, département Mesures Physiques, Université de Rennes 1 en section 61 du CNU.
1998 - 1999 :	ATER à l'IUT de Bézier, département GTR, Université de Montpellier II en section 61 du CNU.
1995 - 1998 :	Thèse de doctorat de l'université de Montpellier II

B Activités d'enseignement

Synthèse des activités

Mes activités d'enseignement sont effectuées à l'Institut Universitaire des Technologies de Lannion depuis ma nomination en tant que maître de conférences, mais j'ai aussi enseigné à l'Institut des Sciences de l'Ingénieur de Montpellier (ISIM) et à l'Institut Universitaire des Technologies de Montpellier dans le cadre de vacations puis en tant qu'ATER.

Depuis ma titularisation dans le département Mesures Physiques de l'IUT de Lannion, les enseignements dispensés lors de cours magistraux, de séances de travaux dirigés et de travaux pratiques représentent la plus grosse partie de mon expérience. Je suis responsable du module de formation d'informatique industrielle pour le département. Ce module, réparti sur les deux années de la formation, regroupe les cours d'électronique numérique, d'architecture des ordinateurs, de la conception de systèmes microprogrammés ainsi que le cours d'acquisition de données. Outre cette responsabilité, je contribue à la formation des étudiants sur l'ensemble de leurs enseignements en électronique par mes interventions en travaux pratiques d'électronique et de traitements du signal, et en assurant des travaux dirigés d'automatique. J'ai aussi assuré l'enseignement d'informatique théorique et assure encore à l'heure actuelle les travaux pratiques d'informatique spécialisée (acquisition de données et contrôle de chaînes de mesures). Dans le cycle de formation des étudiants du département Réseaux et Télécommunications (RT, anciennement GTR), j'ai assuré des enseignements de réseaux et de télécommunications à l'IUT de Montpellier. J'assure à l'heure actuelle un enseignement de programmation micro-contrôleur.

Je suis aussi intervenu dans des cycles de formations supérieures. J'ai ainsi encadré des travaux pratiques de conception réalisés sous forme de "projet industriel de fin d'étude". Ces projets rentrent dans la formation des élèves ingénieurs de 3ème année du département de Microélectronique et Automatique de l'Institut des Sciences de l'Ingénieur de Montpellier. Les sujets abordés ont été l'étude de la compression d'image (algorithme JPEG) et le développement d'une plate-forme de prototypage

Au niveau Master (anciennement DEA) j'ai créé un cours de synthèse de circuits destiné aux étudiants du DEA Systèmes Automatiques et Micro-électroniques, afin de leur permettre d'accéder à une bonne connaissance des outils fréquemment utilisés en micro-électronique. J'ai aussi proposé un cours sur les architectures reconfigurables et sur le système WCDMA pour les étudiants du Master recherche *systèmes intelligents et communicants* de l'université de Cergy-Pontoise. Enfin j'ai créé un cours sur la conception de réseau sur puce, dispensé au master "Systèmes Intelligents Communicants" de l'université de Sousse en Tunisie.

Tout au long de ma carrière, je me suis impliqué dans la formation de stagiaires provenant de différentes formations. Ces encadrements avaient pour but principal de fournir aux stagiaires un environnement de travail de haut niveau et des sujets d'études de qualité. Je suis depuis 2 ans tuteur pédagogique de Ludovic Devaux qui est actuellement moniteur dans le département Mesures Physiques de l'IUT de Lannion.

Récapitulatif des enseignements

Le tableau suivant résume mes activités d'enseignement par cycle d'étude.

Cycle universitaire	Formation dispensée	Département d'enseignement
1^{er} cycle	Electronique et automatique	IUT Mesures Physiques
	Informatique	
	Electronique Numérique	
	Architecture	
	Acquisition de données	
	Réseaux	IUT R&T
	TP Télécom	
	Programmation μ-contrôleur	
$2^{éme}$ cycle	TP conception (PIFE)	ISIM MEA 3
$3^{éme}$ cycle	WCDMA	Master SIIC Cergy
	Architecture reconfigurable	
	Network-on-Chip	Master SiC Sousse
	Synthèse de circuits	DEA SYAM

C Activités de recherche

Parcours de recherche

Maître de conférences depuis septembre 1999 à l'IUT de Lannion (Université de Rennes 1), j'ai intégré dès cette date l'équipe CAIRN de l'IRISA (équipe projet INRIA depuis 2008). Cette équipe est reconnue pour son expertise dans le domaine de la conception de circuits et systèmes intégrés reconfigurables pour les télécommunications mobiles. Le point fort de l'équipe est de regrouper des membres de trois communautés scientifiques : informatique, électronique et traitement du signal. Cette mixité de compétences permet le développement de solutions innovantes dans le domaine des architectures reconfigurables dynamiquement. L'objectif principal de nos travaux est de faciliter la conception de ces systèmes, en proposant des modèles d'architectures associés à des méthodologies de conception qui privilégient l'adéquation entre les applications et les architectures matérielles. Les applications ciblées sont orientées traitement flot de données comme dans les systèmes de télécommunications mobiles, mais aussi les systèmes de transports intelligents, ou encore la cryptographie.

En cohérence avec mes activités antérieures et mes centres d'intérêts, j'ai focalisé mes recherches et participé à la structuration du thème "plates-formes hétérogènes et reconfigurables dynamiquement" de l'équipe. Les objectifs principaux sont de définir de nouveaux systèmes reconfigurables dynamiquement performants et efficaces en énergie, et de proposer des méthodes d'intégration de ces architectures avec d'autres composants (processeur, mémoire, réseau, ...). Lors de ma délégation INRIA de septembre 2006, à septembre 2008, j'ai structuré et renforcé la thématique "gestion des systèmes reconfigurables dynamiquement".

Problématique et fondements scientifiques

Concevoir un système sur puce (SoC – System On Chip) nécessite actuellement de se confronter à deux problèmes majeurs :

La dynamicité requise par les applications, comme la radio logicielle ou la vidéo embarquée, qui nécessitent de supporter des variabilités dans leur exécution en fonction de conditions extérieures (par exemple les performances du canal de transmission). Mais aussi la dynamicité requise par la vitesse d'évolution des applications et/ou des normes et la notion d'*Everyware*, qui sont autant de facteurs nécessitant de supporter de la flexibilité.

La convergence qui dénote le fait d'intégrer tout un panel d'applications dans un seul et même terminal nécessitant la prise en compte de modèles de calcul différents. En effet, ces applications ont des contraintes différentes et requièrent de très fortes capacités de calcul.

Ces constats amènent à la définition d'architectures parallèles hétérogènes intégrant différents opérateurs plus ou moins flexibles, mais performants pour un domaine d'applications.

Dans le même temps, les contraintes de coût, de performance, de faible consommation [PD07], de temps de mise sur le marché et les changements d'usages inhérents aux nouvelles applications embarquées (capteurs intelligents, télécommunications de quatrième génération, cryptographie, ...) poussent au développement d'architectures flexibles et programmables. Ces derniers points, s'ils deviennent essentiels, ne doivent pas en contrepartie affecter l'aspect performance de l'architecture finale. Les nouveaux SoC exhibent donc des propriétés d'exécution dépendant des composants présents physiquement dans la puce finale et offrent de la dynamicité dans l'exécution par le support de la reconfiguration (logicielle ou matérielle) dynamique [AKPD06]. Cependant, la complexité de gestion inhérente à la propriété de dynamicité rend actuellement difficile de tirer pleinement partie des avantages potentiels de ces architectures.

La conception d'un SoC reconfigurable nécessite donc d'implémenter efficacement des **applications dynamiques** sur des **architectures flexibles et parallèles**, ce qui induit une complexité qui n'est pas supportée par les approches de conception classiques. Il est donc nécessaire de repenser ces méthodologies et d'utiliser de nouvelles approches afin de répondre à l'enjeu de la conception des futurs systèmes embarqués. Les difficultés rencontrées pour le contrôle et la conception sur ARD ont amené les concepteurs à virtualiser l'architecture sous-jacente. Cette virtualisation est obtenue par l'introduction d'un système d'exploitation (potentiellement temps réel) qui simplifie le processus de développement des applications et permet l'utilisation de techniques évoluées et dynamiques de gestion de la plate-forme. Dans le contexte de la gestion des ARD, de nouvelles contraintes et de nouveaux services spécifiques doivent être développés.

Mes travaux de recherche se placent dans le cadre de la définition et de la gestion de systèmes reconfigurables dynamiquement. On entend par système l'ensemble architecture, méthodologie de conception et gestion par un OS. Afin d'appréhender l'ensemble des contraintes de la conception à l'utilisation des ARD, j'interviens dans deux thématiques de recherche de l'équipe.

- Technologies et architectures à faible consommation : ce thème porte sur la définition de nouvelles architectures embarquées. Je me place plus particulièrement dans le cadre de la conception de SoC reconfigurables. Mes travaux adressent la problématique de la création d'une plate-forme flexible et reconfigurable dynamiquement.
- Méthodes et conception système : dans cet axe, je travaille à la mise en place d'outils et de méthodes adaptés aux plates-formes reconfigurables. Je m'intéresse plus particulièrement à la définition de services de gestion des architectures notamment sur des aspects placement et ordonnancement de tâches. Des approches de tolérance aux fautes sont aussi explorées dans le contexte des ARD.

Technologies et Architectures à faible consommation

Nous avons proposé une **architecture enfouie reconfigurable dynamiquement** baptisée DART [PSD08] et avons étudié les outils de conception associés. Cette architecture, basée sur des chemins de données reconfigurables organisés en clusters, supporte des performances atteignant 6,2 GOPS (Giga Operation Per Second) par cluster. Dans le même temps, son efficacité énergétique varie, suivant l'application, entre 10 et 40 MOPS/mW. Ce circuit a été réalisé en collaboration avec le CEA et STMicrolectronics sur une technologie 130 nm.

Nos travaux autour de la conception de circuits nous ont amené à étudier le problème des communications pour les futurs systèmes sur silicium utilisant des technologies sub-microniques. En effet, la diminution des tailles de gravure des circuits et l'augmentation des niveaux de métallisation entraînent des phénomènes physiques jusqu'alors négligeables. Il est maintenant indispensable de prendre en compte les phénomènes de bruits et d'erreurs de transmission. Pour cela, nous proposons différentes techniques basées sur l'utilisation d'une logique utilisant un nombre d'états logiques supérieur à deux (MVL – **Multiple Value Logic**) [PKBPS06]. Nous avons alors réalisé une interconnexion supportant de nouveaux protocoles de communication, et permettant d'intégrer des codes correcteurs d'erreurs [PPS06]. Ces travaux ont amené à la définition d'un **codage conjoint** permettant de lutter efficacement contre le *crosstalk* tout en réduisant la consommation du réseau [PCOP09]. Nous étudions actuellement de nouveaux codes permettant d'améliorer la consommation de puissance des communications dans le cadre de liaisons asymétriques [PPS10b].

La flexibilité requise par les applications et par les architectures actuelles nécessite la mise en place de nouvelles infrastructures de communication. Les réseaux intégrés sur puce (**NoC – Network on Chip**) semblent être une bonne alternative pour les communications au sein d'un SoC. En effet, les NoCs permettent d'introduire de la flexibilité dans les communications et permettent d'envisager un grand nombre d'éléments communiquants. La flexibilité est alors fournie par des mécanismes logiciels de routage et de redondance des chemins. Ce modèle est cependant inefficace dans les architectures reconfigurables actuelles. Nous travaillons donc au sein du projet FOSFOR à la définition d'un NoC flexible adapté aux ARD, nommé DRAFT [DCPD09].

Il ressort de nos travaux qu'il y a un besoin de modèles efficaces permettant le développement conjoint des architectures et des outils associés. Ce modèle pour architectures reconfigurables nécessite la définition d'un **langage de description** d'architectures supportant les spécificités inhérentes à la reconfiguration (dynamisme des chemins de données et de contrôle, placement en mémoire, gestion des configurations, ...) [KHK+07]. Nous avons ainsi défini MOZAIC une plate-forme de conception de circuits reconfigurables dynamiquement basée sur xMAML [LPS09a]. Cette plate-forme permet la génération et la définition d'architectures multi-contextes reconfigurables dynamiquement. Les premiers résultats sur ce modèle de reconfiguration nous amènent actuellement à définir l'architecture d'un **FPGA enfoui** (eFPGA) efficace en consommation.

Méthodes et conception système

Dans le projet OverSoC, nous nous sommes orientés vers la définition de services spécifiques qu'un **OS pour Architectures Reconfigurables Dynamiquement** doit offrir pour la gestion *run-time* (donc dynamique) d'une telle architecture [PBG+08]. Pour harmoniser le cadre des réflexions nous avons défini un **modèle de plate-forme** à reconfiguration dynamique vis-à-vis d'un OS (temps réel) qui serait en charge de sa gestion. L'objectif est de proposer un environnement unifié basé sur différents modèles afin d'appréhender les propriétés de chaque composant constitutif du système. Formalisé dans le cadre du projet MARC, nous avons définis un premier modèle d'ARD [PC09]. Nous pensons que l'utilisation de modèles spécifiques pour la partie applicative et pour la partie architecturale permettra de représenter de manière efficace les objets. La problématique majeure concernera donc la création d'équivalence entre modèles et d'assurer un ensemble d'outils permettant une évaluation et une exploration du portage d'une application vers une instance d'architecture (transformations de modèle à modèle).

L'ordonnancement est l'un des services d'OS le plus impacté par la prise en compte d'architectures reconfigurables. En effet, l'aspect dynamique et l'implémentation de tâches matérielles nécessitent un ordonnancement temporel mais aussi spatial des tâches dans l'architecture. Ces travaux correspondent à l'**ordonnancement temps réel** de tâches sur une plate-forme multi-processeurs hétérogènes. En effet, du point de vue de l'ordonnanceur, une architecture reconfigurable est équivalente à un processeur dont les caractéristiques d'exécution varient dans le temps. Nous avons proposé un modèle d'ordonnanceur basé sur les réseaux de neurones [CPS09].

Afin de permettre une gestion dynamique des tâches et de permettre des stratégies d'ordonnancement évoluées, l'équipe CAIRN s'intéresse plus particulièrement à la modélisation et à la réalisation de **communications flexibles** au sein d'une ARD. Dans le cadre du projet FOSFOR, nous intégrerons DRAFT qui supporte un ordonnancement spatial des tâches tout en garantissant l'approvisionnement des tâches en données. La gestion de ce **placement** [ECPS09a], sera couplée avec le réseau afin de fournir un service de communication adapté aux ARD.

Notre participation au GIS ITS (Intelligent Transportation Systems) nous a amené à étudier les besoins des applications du domaine automobile. Dans le cadre du projet CIFAER, nous proposons d'étudier l'apport des architectures reconfigurables dynamiquement pour la **sûreté de fonctionnement**, en étudiant notamment les mécanismes de communications permettant de l'assurer. Ces travaux adaptent des mécanismes de la sûreté de fonctionnement pour des architectures dynamiques [PPD09a]. Ils proposent également une gestion flexible du réseau de communication afin de proposer des mécanismes d'échanges de données sûrs et flexibles [PPD09b].

Je détaille dans la suite de ce document les points forts de ces deux thématiques selon les axes : architecture (chapitre II), interconnexion (chapitre III) et gestion (chapitre IV).

D Perspectives de recherches

Il ne fait actuellement plus aucun doute que les technologies reconfigurables dynamiquement seront un des moteurs du développement à moyen terme. Pour preuve, la sortie prochaine sur le marché du Stratix V, le nouveau circuit commercialisé par la société Altera [55] implémentant ce paradigme. Ainsi les deux constructeurs leader sur le marché vont proposer des solutions dynamiques. Dans le même temps de nouvelles applications et de nouveaux besoins s'expriment et apportent de nouveaux champs d'investigation pour la reconfiguration dynamique (approches de tolérance aux fautes par exemple).
Les perspectives de mon travail de recherche se développeront donc selon les deux axes mis en place tout au long de ces dernières années, et couvriront donc aussi bien des aspects architecturaux que méthodologiques.

Au niveau circuit, l'apparition de nouvelles technologies et des approches "more-than-moore" permettent d'imaginer de nouvelles architectures et de nouvelles perspectives d'intégration. Pour être efficace, ces technologies requièrent de nouveaux paradigmes d'intégration (support des nombreuses défaillances dans les nano-tubes par exemple). De manière à tirer le meilleur partie de ces technologies, des approches fortement parallèles doivent être investies tout en supportant de la flexibilité. Les futurs circuits utilisant le *stacking*[1] vont aussi apporter leurs lots de problèmes de conception, et notamment sur les aspects interconnexion. Le deuxième axe, corrélé au premier, vient de la définition d'outils et méthodes permettant de prendre en compte la nouvelle complexité des architectures envisageables. Ainsi, des approches basées sur l'ingénierie des modèles sont très prometteuses. Enfin, la convergence des technologies permettent d'imaginer des méthodes embarquées de gestion des futures architectures afin de proposer des architectures auto-adaptables. Ce concept nécessite néanmoins encore des recherches sur les aspects système d'exploitation voire même sur le développement de couches intergicielles (selon une approche CORBA par exemple). Ces approches nécessiteront certainement des recherches sur des aspects cognitifs ou bio-inspirés.

1. Les blocs sont empilés verticalement ou horizontalement à côté les uns des autres sur un même substrat.

E Encadrements

J'ai mené une activité d'encadrements continue depuis ma nomination au sein de l'équipe CAIRN. Les encadrements ci-dessous ne représentent que mon activité en tant qu'enseignant-chercheur (période 1999 à maintenant). On remarquera une activité régulière aussi bien en terme de DEA/Master que de thèses. Ne sont pas repris dans ce dossier mes encadrements d'étudiants ingénieurs et autres.

Co-encadrements de thèses

Raphaël David : *Architecture reconfigurable dynamiquement pour applications mobiles.*
Thèse soutenue en juillet 2003,
Taux d'encadrement : 50 %, Directeur de thèse : Olivier Sentieys.
Position actuelle : responsable du laboratoire calcul embarqué du CEA LIST-LETI.

Jean-Marc Philippe : *Intégration des réseaux sur silicium : optimisation des performances des couches physique et liaison.*
Thèse soutenue en novembre 2005,
Taux d'encadrement : 30 %, Directeur de thèse : Olivier Sentieys.
Position actuelle : ingénieur de recherches au CEA LIST-LETI.

Imène Benkermi : *Système d'exploitation temps réel pour architectures reconfigurables dynamiquement à faible consommation.*
Thèse soutenue en janvier 2007,
Taux d'encadrement : 50 %, Directeur de thèse : Olivier Sentieys.
Position actuelle : Ingénieur chez Magneti-Marelli.

Julien Lallet : *Plate-forme générique de modélisation et de conception d'architectures reconfigurables dynamiquement.*
Thèse soutenue en novembre 2008,
Taux d'encadrement : 50 %, Directeur de thèse : Olivier Sentieys.
Position actuelle : Ingénieur de recherche pour Alcatel-Lucent.

Faten Ben-Abdallah : *Étude et optimisation de l'interaction processeurs architectures reconfigurables dynamiquement.*
Thèse soutenue en octobre 2009,
Taux d'encadrement : 50 %, Directeur de thèse : Olivier Sentieys.
Thèse en co-tutelle avec l'ENIT de Tunis
Position actuelle : Enseignante associée à l'Ecole Nationale d'Ingénieur de Sousse, Tunisie.

Manh Pham : *Plate-forme de calcul générique pour véhicule intelligent.*
Thèse à soutenir le 15 décembre 2010,
Taux d'encadrement : 70 %, Directeur de thèse : Didier Demigny.

Ludovic Devaux : *Interconnexion flexible pour architecture reconfigurable dynamiquement.*
Thèse à soutenir en 2011,
Taux d'encadrement : 70 %, Directeur de thèse : Didier Demigny.

Antoine Eiche : *Ordonnancement temps réel pour architectures hétérogènes reconfigurables à partir de structure de réseaux de neurones.*
Thèse à soutenir en 2011,
Bien que non encadrant je suis impliqués dans le déroulement de ces travaux, Directeur de thèse : Olivier Sentieys.

Istas Pratomo : *Supports dynamiques de communications : vers la définition d'un NoC adaptatif dans un SoC reconfigurable.*
Thèse à soutenir en 2013,
Taux d'encadrement : 70 %, Directeur de thèse : Olivier Sentieys.

Encadrements de Master

Faten Ben-Abdallah : *Couplage d'un cluster de DART dans un processeur VLIW.*
stage de DEA Systèmes de communication en 2003, ENIT, Tunis,
Taux d'encadrement : 100%.

Julien Lallet : *Vers des architectures à reconfiguration dynamique préemptive.*
stage de DEA de l'université de Nancy I en 2005,
Taux d'encadrement : 100 %.

Lalit Garg : *The definition of the Compilation tools for Re-configurable platform.*
stage de DEA STIR de l'université de Rennes 1 en 2006,
Taux d'encadrement : 50 %, Co-encadrant : Daniel Chillet.

Daoud Karalolah : *Contrôleur intelligent et communiquant pour la gestion répartie de tâches reconfigurables sur SoC.*
stage de Master SIIC de l'ENSEA (Cergy-pontoise) en 2006,
Taux d'encadrement : 50 %, Co-encadrant : Didier Demigny.

Umer Farooq : *Etude d'un réseau flexibles pour SoC reconfigurable.*
stage de Master STI de l'université de Nice en 2007,
Taux d'encadrement : 50 %, Co-encadrant : Daniel Chillet.

Ludovic Devaux : *Etude et implémentation du service de communication d'un OS pour la gestion d'un réseau flexible au sein d'un SoC reconfigurable.*
stage de Master STIR de l'université de Rennes 1 en 2008,
Taux d'encadrement : 50 %, Co-encadrant : Daniel Chillet.

Sana Ben Sassi : *Génération automatique de NoC flexible.*
stage de Master SIC de l'université de Sousse en 2009,
Taux d'encadrement : 100 %.

Adnan Akhtar : *Modèle de simulation et de vérification de MPSOC reconfigurable dynamiquement.*
stage de Master *System on Chip* du KTH Stockholm en 2010,
Taux d'encadrement : 100 %.

Ferhat Abbas : *Etude des langages et modèles pour la description de systèmes sur puce.*
stage de Master STIR de l'université de Rennes 1 en 2010,
Taux d'encadrement : 50 %, Co-encadrant : Daniel Chillet.

Surya Nataraja : *Gestion dynamique d'un NoC flexible.*
stage de master *Computer Engineering* de TU Delft en 2010,
Taux d'encadrement : 50 %, Co-encadrant : Daniel Chillet.

F Rayonnement scientifique

Relations scientifiques

Au cours de mon exercice en tant qu'enseignant-chercheur, j'ai participé et monté des collaborations autour de projets et de contrats de recherches. Autour de ces projets, j'ai bien évidemment renforcé mes axes de recherches personnels, mais ces travaux ont aussi permis de développer de nouveaux axes de travail. La figure 1-1 montre de manière synthétique la chronologie des différents projets dans lesquels je suis intervenu, ou interviens encore.

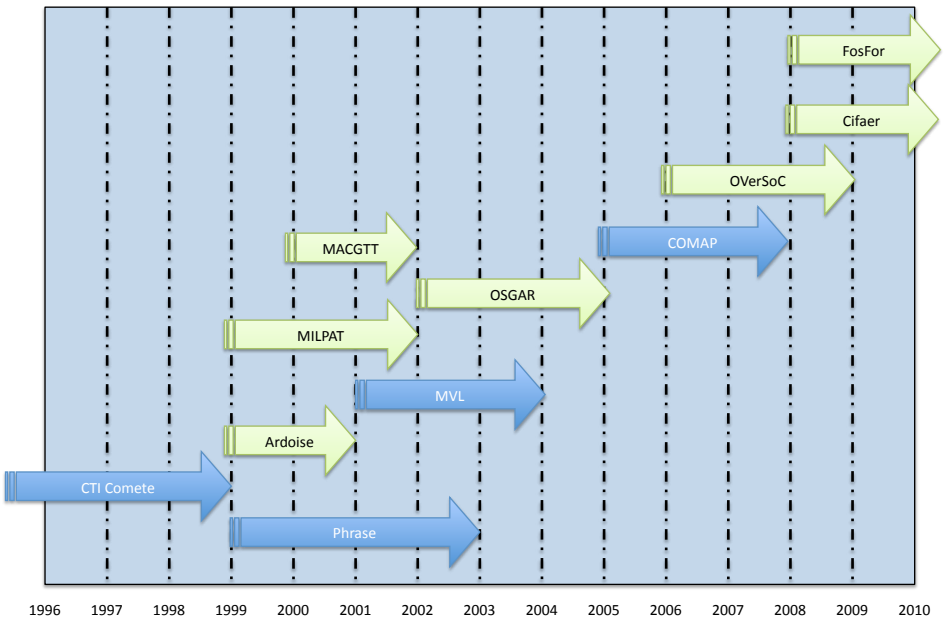


Figure 1-1 – **Participation à des projets et contrats nationaux et internationaux**.

Participation à des contrats de recherche nationaux et internationaux

- ***CTI COMETE*** : *Codesign : Conception conjointe logiciel-matériel.* 1995-1999
  Ce contrat de recherche du CNET avait pour objectif l'étude de méthodologie de conception conjointe logiciel-matériel autrement appelé Codesign.
  Ces travaux ont permis de proposer des méthodes permettant de spécifier un produit (sans

faire intervenir les logiciels et les matériels qui le constitueront), puis de séparer harmonieusement le logiciel du matériel, pour ensuite les construire et les valider en parallèle.

- ***PHRASE*** : *Reconfigurabilité et processeurs VLIW dans les architectures parallèles hétérogènes.* 1999-2003
  Ministère de l'Industrie - programme RECMES. En collaboration avec l'Université de Bretagne Occidentale et *STMicroelectronics* (Grenoble et San José).
  Il s'agissait dans ce projet de proposer une architecture fortement parallèle et hétérogène et de proposer les outils de conception associés permettant de tirer parti des performances potentielles du matériel.

- ***MVL*** : *Research on MVL/SUS-LOC architecture for telecommunication applications.* 2001-2004
  Joint University-Industry Research and Licensing Agreement with EDO Inc, USA.
  L'objet principal de ce contrat visait à démontrer l'intérêt d'un brevet de la société EDO permettant l'implémentation aisée de logique multi-valuée. Nous devions concevoir un cœur de processeur complet utilisant cette technologie. Les résultats de ce projet ont démontré l'intérêt de la logique MVL mais n'ont pas permis d'aboutir à un réel démonstrateur, l'intégration de ces technologies restant complexe. Cependant, cette étude a amené les travaux autour de l'utilisation de la logique MVL pour concevoir une interconnexion fiable et faible consommation.

- ***CoMAP*** : *Co-design Of Massively Parallel Embedded Processore Architectures.* 2005-2008
  Programme P2R. En collaboration avec l'Université de Bretagne Occidentale, l'ENST Bretagne, l'Université de Erlangen-Nuremberg (Allemagne) et l'Université de Dresde (Allemagne).
  La méthode que nous utilisons dans ce projet est fondée sur la représentation abstraite d'architectures, sur laquelle on construit un cœur d'outils génériques que l'on sait étendre. Puis, on décrit les applications sur ces outils avec des transformations de complexité variée. La représentation abstraite est alors utilisée d'une part pour spécialiser les outils et d'autres part pour réaliser l'architecture parallèle sous-jacente.

Participation à des projets nationaux

- ***ARDOISE*** *: Architecture Reconfigurable Dynamiquement Orientée Image et Signal Embarqué.* 1999-2001
  Action incitative inter GDR ISIS et GDR ARP "Systèmes Reconfigurables".
  Le projet ARDOISE a réuni six équipes de recherche (ayant une expérience reconnue en architecture des systèmes temps réel pour le traitement d'images) qui ont coopéré à la spécification d'une architecture à reconfiguration dynamique. Cette action a mené à la conception de la première architecture reconfigurable dynamiquement européenne.

- ***MILPAT*** *: Méthodologie et développement pour les InteLlectuals Properties (IP's) pour Appli-cations Télécom.* 1999-2001
  Projet labellisé par le RNRT (Réseau National de Recherche en Télécommunications). En collaboration avec France Telecom R&D (Grenoble) et le LESTER de l'Université de Bretagne Sud (Lorient).
  Les objectifs étaient de formaliser la démarche de conception d'un IP au niveau comportemental (composant virtuel), de spécifier des applications de traitement du signal à base d'IPs et de réaliser une étude de cas pour la conception d'applications télécom. Dans ce cadre nous avons proposé un environnement logiciel supportant l'aide à la spécification et au paramétrage de ces composants virtuels.

- ***MACGTT*** *: Méthode d'Aide à la Conception Globale des Terminaux de Télécommunications.* 2000-2002
  Programme Télécom CNRS. En collaboration avec l'*I3S* - Université de Nice Sophia Antipolis et le *LESTER* - Université de Bretagne Sud.
  L'objectif principal de ce projet était la proposition d'une méthodologie globale de conception système. L'approche que nous avons proposée, consiste à décomposer la conception système en deux phases. La première, basée sur des techniques d'estimation, a pour objectif de réduire l'espace de conception. Dans une deuxième phase, une exploration automatique et/ou interactive de cet espace restreint est réalisée par un algorithme de partitionnement/ordonnancement.

- ***OSGAR*** : *Outils de Synthèse Générique pour Architectures Reconfigurables.* 2002-2005
  Projet labellisé par le RNTL (Réseau National des Technologies Logicielles). Les partenaires du projet OSGAR étaient le CEA, la société TNI-Valiosys et l'Université de Bretagne Occidentale.
  L'objet de ce projet était de travailler sur des technologies de synthèse générique (au sens multi-cibles) pour des architectures reconfigurables. Nous avons proposé des outils capables d'effectuer des opérations de synthèse de haut niveau afin de prototyper rapidement le portage d'applications sur des structures de traitements reconfigurables.

- ***OverSoC*** *: Outil de Validation et d'Exploration de SoC Reconfigurable.* 2006-2009
  Programme labellisé par l'ANR dans le cadre de l'Action de Recherche Amont Sécurité Systèmes embarqués et Intelligence Ambiante. En collaboration avec ETIS - Université de Cergy-Pontoise et le LISIF - Université Pierre et Marie Curie.
  Ce projet avait pour objectif de développer une méthodologie permettant de modéliser les services d'un système d'exploitation temps réel destiné à gérer une plate-forme matérielle disposant de ressources reconfigurables. Les résultats de ce projet sont un flot d'exploration incluant le système d'exploitation, ainsi qu'un outil facilitant la mise en œuvre de ce flot.

- ***FOSFOR*** *: Flexible Operating System FOr Reconfigurable platform.* 2008-2010
  Programme labellisé par l'ANR Architectures du Futur. Les partenaires sont le LEAT - Université de Nice Sophia Antipolis, ETIS - Université de Cergy-Pontoise et Thales TRT.
  Le projet vise à proposer un RTOS flexible, distribué, et proposant une interface homogène du point de vue de l'application, pour permettre la gestion d'une plate-forme hétérogène intégrant des composants reconfigurables dynamiquement. Dans le cadre de ce projet nous nous intéressons plus particulièrement à la définition du service de communication de l'OS ainsi qu'au support matériel de cette communication au sein de la zone dynamique.

- ***CIFAER*** *: Communications Intra-véhicule Flexibles et Architecture Embarquée Reconfigurable.* 2008-2011
  Programme labellisé par l'ANR Architectures du Futur et le Pôle Automobile Haut de Gamme. En collaboration avec l'IETR - Université de Rennes 1, IREENA - Université de Nantes, la société GEENSYS et la société ATMEL.
  L'activité de recherche menée au sein du projet CIFAER porte sur la définition d'une architecture reconfigurable innovante pour le domaine de l'automobile. Le projet s'intéresse à la définition du cœur numérique (processeur généraliste associé à une zone reconfigurable) de l'ECU ainsi qu'au support d'interfaces de communication flexibles (lien radio et courant porteur). Cette architecture supportera alors des contraintes de sûreté de fonctionnement requises par le domaine d'application. Je suis le responsable scientifique de ce projet pour l'équipe CAIRN.

Participation à des projets régionaux

Je suis responsable du montage et du suivi scientifique des deux projets régionaux suivant :

- ***Sehraiv*** *: Self-Healing Reconfigurable Architecture for Intelligent Vehicles* 2007-2009
  Projet financé par la région Bretagne et le conseil général des Cotes d'Armor.
  Dans ce projet, nous nous proposons de démontrer l'intérêt des nouvelles technologies reconfigurable dynamiquement dans le domaine de l'automobile et plus généralement dans le domaine de la sûreté de fonctionnement. L'utilisation de calculateurs reconfigurables dynamiquement a un double intérêt : la réduction du nombre de calculateurs réduisant ainsi les coûts de mise en œuvre, et le support efficace des approches classiques et industrielles de la sûreté de fonctionnement.

- ***MARC*** *: Modélisation d'Architectures Reconfigurables et de leur Contrôle.* 2010
  Programme "Défis scientifiques émergents" labellisé par l'Université de Rennes 1.
  L'objectif de ce projet consiste à définir une méthodologie d'exploration d'un système (à la fois matériel et logiciel) en se basant sur des techniques de typage de modèles afin de faciliter la réutilisation et l'interopérabilité entre les différentes composantes d'un même système. Il s'agit de définir les modèles de haut-niveau (méta-modèles) des différents éléments composants le système complet (application, architecture, operating system) en adéquation avec les

spécificités de ces éléments, mais aussi en vue de leur utilisation et de leur transformation pour la génération d'une chaîne de conception et d'exploration adaptée. La bijectivité des modèles devra être assurée afin de permettre des phases d'exploration depuis l'application vers l'architecture ou inversement.

Activités d'expertises

En tant que maître de conférences j'ai participé aux commissions de spécialistes de la 61ème section de l'université de Rennes 1 en tant que membre élu (2000-2008), et de l'université de Cergy-Pontoise en tant que membre nommé (2004-2008). Je participe depuis régulièrement aux commissions de sélection de ces deux universités.

Mes activités de recherche m'ont amené à être expert pour l'ANR lors des appels à projets Architecture du futur (2007) et ARPEGE (2009 et 2010).

Enfin je suis membre du comité de programme de différentes conférences parmi lesquelles *DASIP, DTIS, ERSA, IEEE FPL, IEEE NewCAS, SPL*, et relecteur pour différentes revues et conférences comme *IEEE Transactions on VLSI*, *International Journal on Reconfigurable Circuits*, IET Computer and Design, ACM Transactions on Embedded Computing Systems, transaction on VLSI design ou ACM/IEEE DAC, GRETSI.

Valorisation de la recherche et transfert technologique

Mes travaux de thèse étaient soutenus par un contrat de longue durée du Centre National d'Etude des Télécommunications (CNET). Outre les présentations d'avancement des travaux, j'ai installé notre environnement de prototypage dans les locaux du CNET à Meylan et collaboré à sa mise en place à l'Université de Porto Alegre (UFGRS) au Brésil. Cette université, par l'intermédiaire de l'équipe de M. Fernando Moraes, a travaillé sur l'environnement que j'ai dévellopé lors de ma thèse et a obtenu des résultats très intéressants, notamment pour l'intégration de réseaux de neurones.

Mon activité de valorisation présente aussi des aspects transferts de connaissances lors d'une intervention pendant l'école d'été CNRS "ARCHI'03" pour laquelle j'ai réalisé un cours sur les architectures reconfigurables. J'ai été membre du comité d'organisation de "ARCHI'09" à Trébeurden. Cette diffusion à aussi pris place lors de mes enseignements dans les Masters de l'université de Cergy-Pontoise et de Sousse en Tunisie ainsi que lors de séminaires des différents GDR auxquels je participe.

J'ai aussi réalisé deux court séjours à l'Université Laval à Québec dans le cadre de l'équipe

associée entre le LRTS et l'IRISA. J'ai réalisé à ce titre un séminaire sur le placement de tâches sur architectures reconfigurables dynamiquement.

Je suis organisateur d'une session spéciale "reconfiguration dynamique" pour la conférence DASIP 2011, et je suis membre du comité d'organisation de SYMPA 2011 à S^t Malo.

Participation aux organismes d'animation de la recherche

Je participe activement aux instances d'animation de la communauté au travers des réunions des GDR et d'autres structures d'animation. Ces lieux d'échanges permettent régulièrement l'émergence d'idées nouvelles et de projets nationaux.

- Membre de l'Equipe Projet Multi Laboratoire *POMARD : Projet Outils, Méthodes et Architectures pour la Reconfiguration Dynamique* du département STIC du CNRS. Cette action a amené le montage du projet OverSoC. 2003-2004.
- Participation aux *Actions Spécifiques* du RTP Systèmes sur puce du CNRS. 2002-2003. Participation à plusieurs groupes de travail :
  - AS Architecture reconfigurable dynamiquement,
  - AS Adéquation système d'exploitation - architecture,
  - AS intergiciel pour la radio-logicielle.
- Membre du GdR-PRC ISIS (Information Signal ImageS), Thème C "Adéquation Algorithmes Architectures".
- Membre du GdR-PRC SoC-SIP (System On Chip - System In a Package), depuis sa création. Participation aux groupes de travail :
  - Architectures Reconfigurables,
  - Logiciels Embarqués et Architectures Matérielles,
  - Consommation et Energie dans les SOC/SIP.
- Membre du GdR-PRC ARP (Architectures Réseaux et Parallélisme), groupe de travail "Architectures Spécialisées" puis du GDR ASR (Architecture, Systèmes, Réseaux).
- *GIS ITS* : Groupement d'Intérêt Scientifiques sur les Systèmes de Transports Intelligents. Dans le cadre de l'aménagement du territoire, la région Bretagne a monté un pôle de compétence autour des ITS auquel je participe.
- Je suis Membre de l'IEEE depuis 1999, et du club EEA, depuis Janvier 2000.

Autres activités

D'une manière générale je participe à la "vie" de l'IUT et de la recherche au travers de mon implication dans diverses instances de gouvernance.

- Membre élu du conseil de laboratoire de l'IRISA de 2004 à 2008.
- Membre élu du comité de centre INRIA Rennes Bretagne Atlantique depuis 2008.
- Membre élu du conseil de direction de l'IUT de Lannion de 2005-2007.
- Membre élu du conseil d'institut de l'IUT de Lannion depuis 2004-2007.
- Responsable réseau du département Mesures Physiques de l'IUT de Lannion depuis 2003.

G Bibliographie

Je ne présente dans cette section qu'une vue synthétique de mon activité de publication. La liste détaillée est présentée à la fin de ce document dans la section bibliographie. On notera une activité constante avec un effort particulier sur les revues depuis 2008. Le nombre de publications de ces dernières années est le fruit de mes deux années de délégation INRIA, qui m'ont permis de mettre en œuvre mon projet de recherche et dont les résultats commencent à être matures.

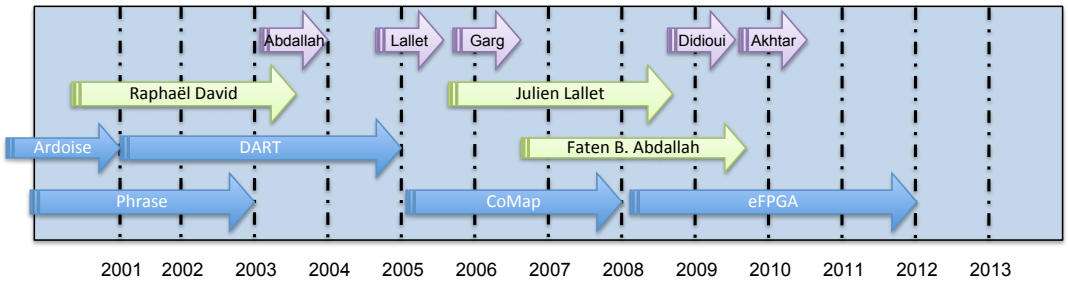
	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Revues Int.													1	3	4
Revues Nat.										1		1		1	1
Chapitre de livre		1	1	1			1		1						1
Conf. Int.	2		1	2		2	6	1	1	3	3	3	4	8	7
Conf. Nat.						2	1	3		1		1		3	
Colloque	1	2	1				1		1		1	1		1	

J'ai actuellement, deux articles de revue et trois articles de conférence en soumission.

Chapitre

2 Architectures Reconfigurables Dynamiquement

Résumé : Ajouter de la reconfiguration dynamique dans les systèmes permet d'optimiser les coûts d'utilisation, d'améliorer les performances et potentiellement la consommation d'énergie. Malheureusement, tous ces bénéfices ne sont atteignables qu'au prix d'un effort significatif de conception. Il est donc important de concevoir des architectures efficaces, mais aussi les flots de conception permettant de manipuler ces architectures de manière simple et intuitive. Je présente dans ce chapitre mon activité autour de la conception d'architectures reconfigurables dynamiquement et des méthodes associées. Le graphique suivant présente chronologiquement les différents projets ainsi que les étudiants de masters et les thèses impliqués dans cette thématique.



Sommaire

A Introduction

Les systèmes matériels actuels doivent, dans un encombrement très faible, supporter des puissances de calcul de plus en plus grande, tout en consommant de moins en moins d'énergie. Les développements actuels visent par ailleurs à intégrer de plus en plus de services différents dans un même produit, on parle alors de convergence. Cette propriété nécessite la mise en place de systèmes hétérogènes, hautes performances et flexibles.

De nouvelles approches dans la conception de produits micro-électroniques émergent, et il est clair qu'une des solutions est la reconfiguration dynamique. L'idée de base de ces architectures est d'offrir aux concepteurs la flexibilité d'une architecture programmable et les performances temporelles d'un circuit dédié. Il existe deux pistes permettant d'envisager une utilisation efficace de ce paradigme d'exécution. La première consiste à concevoir des tableaux très réguliers d'éléments simples et flexibles. Cette structure est parfaitement adaptée à une intégration dans les futures technologies. C'est la structure classique des FPGA du commerce, dont le leader Xilinx propose plusieurs familles de circuits (comme le Virtex 6 [90]). Cependant, d'autres architectures commerciales existent comme l'ATMEL AT40k40 [6], ou vont exister comme le Stratix V de Altera [55].

La deuxième piste vient de l'utilisation de structures de plus gros grain permettant un réarrangement de chemins de données complexes (au niveau arithmétique). La recherche dans ce domaine a été très intense dans les années 2000 et a vu l'apparition de nombreuses architectures [3][20]. Ce deuxième type d'architectures permet d'optimiser le processus de reconfiguration en diminuant la taille des *bitstreams*[1]. Cette optimisation permet alors de réduire le temps du processus de reconfiguration ainsi que sa consommation d'énergie.

Les travaux autour de l'architecture *DART* s'inscrivent directement dans ce contexte. La solution proposée repose sur la définition d'un modèle d'architecture "gros grain" offrant de meilleures performances que les FPGA pour des applications de télécommunications embarquées. Notre solution prend en compte, à tous les niveaux de la conception, la contrainte de faible consommation requise par ces systèmes, tout en supportant la puissance de calcul requise.

Si développer des architectures "gros grain" permet d'optimiser certains aspects de l'architecture, il en coûte relativement cher en terme de flexibilité. Les architectures "grain fin", de type FPGA, offrent alors encore de belles perspectives d'intégration. La difficulté de conception des architectures et des outils de synthèse associés nous a amené à étudier un langage de spécifications pour architectures reconfigurables dynamiquement. Dans ce contexte nous avons proposé, avec l'environnement Mozaïc, un ensemble cohérent méthodologie-architecture pour des architectures reconfigurables dynamiquement. Nous avons alors dans ce contexte proposé une architecture de FPGA enfoui (eFPGA) qui a pour objectif de réduire le coût de la reconfiguration d'un FPGA et de supporter des mécanismes de gestion efficaces permettant d'envisager une vraie dynamicité dans l'exécution des applications.

1. Ensemble des bits permettant la configuration d'un circuit reconfigurable.

B L'architecture DART

Contexte
Cette partie porte sur des travaux qui ont été initiés dans le cadre de la thèse de Raphël David [26] sur la période 2000-2003. L'objectif est d'optimiser une architecture reconfigurable dynamiquement en consommation énergétique tout en conservant des niveaux de puissance de calcul compatible avec le domaine applicatif. Le projet FRESH, en collaboration avec le CEA, sur la période 2003-2005 a permis de réaliser physiquement le circuit.

L'architecture DART a été conçue dès le départ afin d'optimiser son efficacité énergétique (EE). Cette efficacité peut être définie comme le nombre d'opérations effectuées par mW de consommation. Cette grandeur permet alors de représenter la puissance de calcul de l'architecture ainsi que sa consommation énergétique.

Le premier paramètre à optimiser est très certainement la tension d'alimentation puisque la puissance a une relation quadratique avec cette grandeur. La difficulté vient du fait que baisser la tension d'alimentation amène à réduire les fréquences de fonctionnement. Afin de palier ce problème, l'exploitation du parallélisme est une solution. Ce parallélisme peut être défini à plusieurs niveaux. Premièrement, les applications du domaine multimédia utilisent différentes tailles de données (par exemple 8 bits en vidéo et 12 bits pour de l'audio). Afin de supporter ces différentes tailles de codage et d'optimiser les ressources, des opérateurs *Sub-Word Processing* (SWP) [34] supportant des tailles de données de 8, 16 et 32 bits peuvent êtres conçus. Ces opérateurs permettent alors d'augmenter le parallélisme de données en fonction de l'application. Deuxièmement, le parallélisme d'instructions (ILP) est inhérent aux applications de calcul intensif. Son exploitation est simple puisqu'il ne requière que l'implémentation d'opérateurs indépendants travaillant en parallèle, c'est le principe des architectures *Very Long Instruction Word* (VLIW). Finalement, le parallélisme d'applications représente le nombre de tâches indépendantes exécutables en parallèle. Cette contrainte amène à la définition d'architectures "clusterisées", chaque cluster supportant une application, ou pouvant être couplé pour supporter des tâches plus complexes. L'exploitation efficace du parallélisme permet des optimisations avancées de la consommation. Par exemple, une allocation efficace des tâches permet de mettre en mode veille (voire d'éteindre) certaines parties de l'architecture [84], ou d'introduire d'autres mécanismes comme des horloges gardées (*clock gating*) [17].

Dans une architecture reconfigurable, le contrôle et la distribution du *bitstream* ont un impact sur la consommation d'énergie. De ce fait, la taille des bitstreams ainsi que la fréquence de reconfiguration doivent être optimisées. La taille des bitstreams est optimisée en réduisant le nombre de composants à configurer. C'est la différence majeure entre les architectures grain-fin et les architectures gros-grain. Si il existe de la redondance dans le chemin de données alors la taille du bitstream peut aussi être réduite par l'utilisation du concept de *Single Configuration Multiple Data* (SCMD), où l'idée est de configurer plusieurs éléments avec le même mot de

configuration. Cette idée a été utilisée pour la première fois dans le FPGA Xilinx XC6200 [16] [2]. Le principe était d'utiliser des "wildcard" dans les adresses de configuration afin de programmer plusieurs cellules en même temps. La règle des 80/20 [86] dénote le fait que 80% du temps d'exécution est dû à 20 % du programme de l'application. Cette portion de code est souvent constituée de boucles très répétitives et nécessitant beaucoup de calculs (nested loops). Dans ce type de code, un calcul identique est répété un grand nombre de fois. Reconfigurer ce type de calcul permet alors des gains en terme de temps d'exécution. Le reste du code, environ 80%, est difficile à optimiser car il représente essentiellement du contrôle changeant à chaque cycle. Des mécanismes de reconfiguration adéquats sont alors nécessaires.

Réduire le coût des accès aux données est un exercice tout aussi délicat que critique dans un contexte de faible consommation. Il faut à la fois limiter le nombre d'accès aux mémoires de données et le coût de ces accès. Une des techniques permettant de réduire le nombre d'accès est de chainer les opérateurs entre eux. Ce pipeline d'opérations évite alors le stockage temporaire des résultats intermédiaires. Une autre stratégie est d'offrir du stockage local afin d'augmenter la localité des références. La diffusion un-vers-tous (*broadcast*) permet aussi d'optimiser les accès mémoire en permettant la diffusion d'une même donnée à tous les consommateurs qui en ont besoin en une seule lecture. On peut aussi optimiser ce concept en introduisant des registres permettant un stockage local des données afin de créer des chaînes à retard, pour des accès à des données de type vecteur par exemple. Enfin, une hiérarchie mémoire adaptée est nécessaire. Ainsi, en plus des petites mémoires locales, une mémoire globale permettra le stockage de masse des données mais devra être contrôlée de manière efficace.

L'association de tous ces principes a amené à la première définition de l'architecture DART.

B.1 Architecture DART

Cette architecture a été conçue de manière hiérarchique. Au niveau système, DART est un ensemble de clusters indépendants qui ont accès à un espace d'adressage commun. Un contrôleur général permet alors de gérer l'ensemble de l'architecture en supportant, par exemple, un OS pour l'assignation des tâches à exécuter. Cette vision correspond à une architecture reconfigurable autonome. Une autre vision consiste à ne considérer qu'un seul cluster, qui travaille alors comme un accélérateur matériel dans un RSoC.

B.1-1 Architecture d'un cluster DART

Un cluster DART (figure 2-1) est composé de *datapaths reconfigurables* (DPR) interconnectés au travers d'un réseau de bus segmentés. A ce niveau, se retrouve la mémoire de configuration ainsi que son contrôleur. Toutes les primitives de calcul accèdent au même espace d'adressage.

2. On notera que cette architecture est considérée comme la première supportant la reconfiguration dynamique et partielle. De nombreux concepts étaient déjà présents dans ce circuit, qui n'a malheureusement pas connu le succès commercial attendu et disparaitra donc.

DART a été conçue comme une architecture orientée plate-forme dans laquelle une zone connectée au réseau permet d'ajouter de la logique dédiée requise par l'utilisateur. On peut, par exemple, y intégrer un cœur de FPGA enfoui afin de supporter efficacement des opérations au niveau bit.

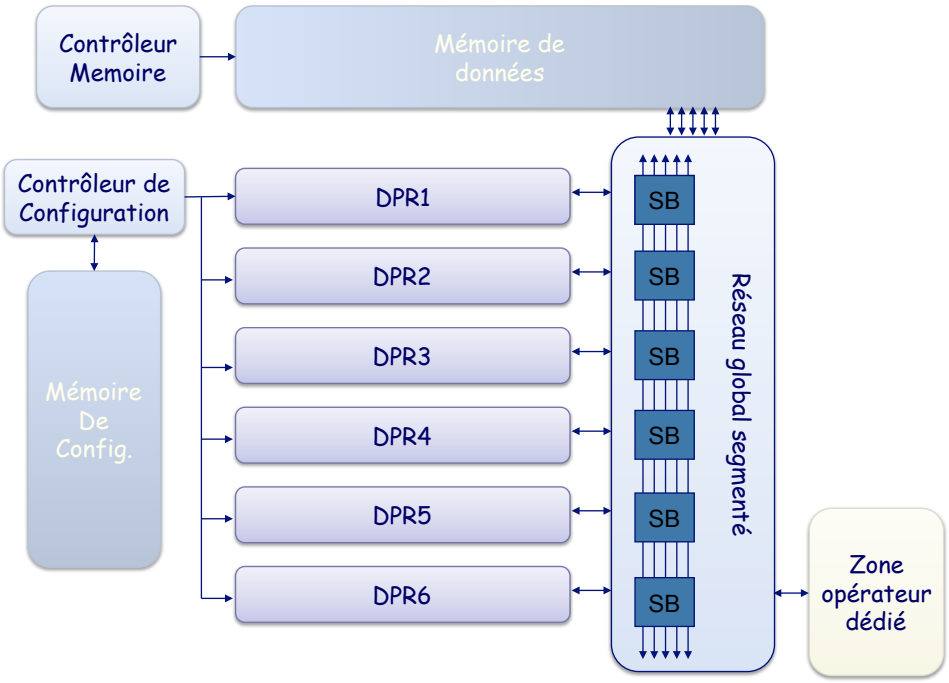


Figure 2-1 – **Architecture d'un cluster de DART**. *Construit autour d'un réseau de bus segmentés, un cluster comprend six* datapaths reconfigurables *(DPR), une mémoire de configuration et son contrôleur, et une zone optionnelle dédiée. La mémoire de données de ce niveau représente le plus haut niveau de la hiérarchie.*

Le réseau de bus permet de considérer les différents DPR de manières indépendantes afin de traiter des tâches individuelles, ou d'interconnecter les ressources sous forme de pipeline pour supporter des tâches plus complexes. Ce réseau permet d'adapter dynamiquement l'interconnexion des DPR et permet aussi l'accès à la mémoire de données de ce niveau. L'organisation hiérarchique de DART permet de distribuer le contrôle afin d'optimiser la consommation d'énergie, notamment en réduisant les interconnexions entre les contrôleurs et les chemins de données. La tâche principale du contrôleur de reconfiguration est de gérer la reconfiguration des DPR. Ce contrôleur intègre la notion de SCMD, permettant de réduire le nombre d'accès mémoire et d'optimiser le temps de reconfiguration.

B.1-2 Architecture des chemins de données

Les DPR sont les primitives de calcul arithmétique de DART (figure 2-2). Ils sont organisés autour de quatre unités fonctionnelles (UF décrites dans la section B.1-3), incluant chacune un registre de pipeline. Ces composants sont interconnectés par un réseau multi-bus flexible.

Toutes les UF, deux multiplieur/additionneur (MUL 1 et 2) et deux unités arithmétique et logique (ALU 1 et 2), supportent le mode SWP et sont reconfigurables dynamiquement.

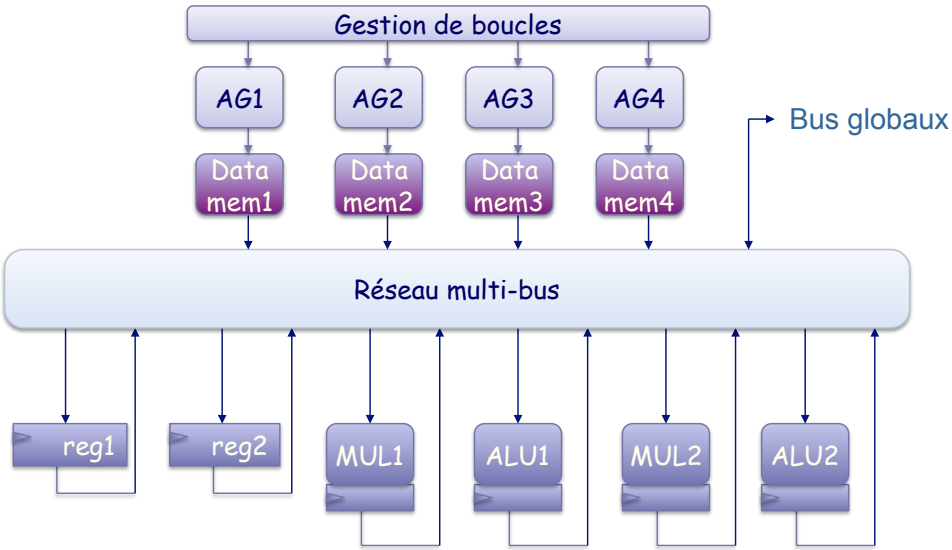


Figure 2-2 – **Architecture d'un chemin de donnée reconfigurable (DPR)**. *Construit autour d'un réseau multi-bus de type crossbar, quatre unités fonctionnelles travaillent sur des données stockées localement dans de petites mémoires SRAM.*

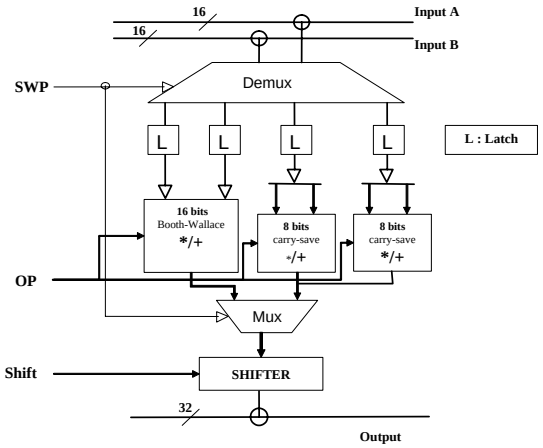
Les UF travaillent sur quatre mémoires SRAM locales (Data mem) contrôlées par des générateurs d'adresses (AG – Address Generator) locaux. Ces unités ont en charge le contrôle de l'accès aux mémoires, et doivent fournir les données aux UF. Afin de supporter un large spectre d'applications, ces générateurs supportent un grand nombre de motifs d'accès mémoire (bit-reverse, modulo, pre/post increment, ...). Les AG sont de petits cœurs RISC optimisés capables d'adresser les petites SRAM locales. Chaque AG peut être arrêté s'il n'intervient pas dans un calcul pour des raisons de consommation. Les générateurs d'adresses partagent un gestionnaire de boucles imbriquées. Grâce à ce composant, il est possible de gérer jusqu'à quatre niveaux d'imbrication de boucles sans aucun cycle d'exécution supplémentaire.

En plus des mémoires locales, deux registres supplémentaires sont intégrés dans les DPR afin de pouvoir créer des lignes de retard, permettant de partager des données dans le temps sans refaire d'accès mémoire.

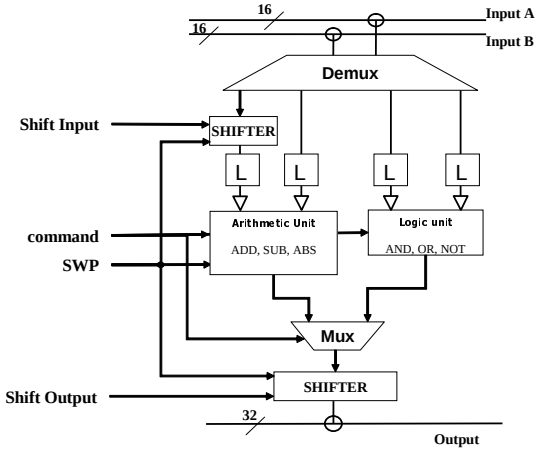
Le réseau multi-bus est entièrement connecté, constituant alors un *crossbar*. Ce second niveau d'interconnexion offre donc une très grande flexibilité, mais reste limité en terme de complexité car très localisé au sein des DPR. Ce réseau permet alors de créer n'importe quel chemin de données entre les différents éléments du DPR, et supporte l'optimisation de n'importe quel schéma de calcul. Le réseau multi-bus permet en outre un accès simultané à n'importe quelles mémoires, ou supporte la diffusion de type *multicast*. Enfin, c'est lui qui assure la connexion du DPR au réseau global pour transférer des données entre les DPR ou avec la mémoire globale du système.

B.1-3 Architecture des Unités Fonctionnelles

Pour concevoir une architecture efficace, la conception des unités fonctionnelles est primordiale. DART est basée sur deux unités différentes supportant les traitements SWP (au vu des tailles de données rencontrés dans les applications actuelles). Les UF sont donc conçues pour travailler par défaut sur des données de 16 bits, et supportent des données codées sur 8 bits. La première unité implémente un multiplieur-additionneur. Concevoir un multiplieur faible consommation est difficile mais est un domaine bien connu. Pour des mots de 16 bits, une des architectures la plus efficace utilise l'algorithme de *Booth-Wallace* [56]. Pour l'implémentation des traitements SWP, notre étude a montré qu'implémenter trois multiplieurs, un de 16 bits et 2 de 8 bits (avec une architecture *carry-save*), permettait une meilleure optimisation du compromis surface-temps-énergie par rapport à d'autres solutions. Cette approche permet, de plus, de gérer l'énergie en arrêtant le fonctionnement des blocs non utilisés dans un calcul. L'unité fonctionnelle, représentée sur la figure 2-3a, inclut la saturation pour les résultats signés. Une mise à l'échelle du résultat est réalisée grâce au décaleur (*shifter*) en sortie de la multiplication. L'arbre de Wallace est réutilisé pour réaliser l'additionneur de cette UF.



(a) Unité fonctionnelle mul/add.



(b) Unité arithmétique et logique

Figure 2-3 – **Architecture des unités fonctionnelles de DART**. *Les deux unités fonctionnelles supportent les traitements SWP et intègrent des décaleurs pour le recadrage des données. La saturation est implémentée dans le même cycle d'exécution que l'opération elle-même. Les entrées des UF sont synchronisées afin de réduire l'activité des données et ainsi d'optimiser la consommation de ces opérateurs.*

La deuxième UF, décrite sur la figure 2-3b, implémente une unité arithmétique et logique. Elle supporte les opérations d'additions, de soustractions et de logiques classiques (ET, OU, Ou exclusif, ...). Elle est construite autour d'un additionneur de *Sklansky*, qui a été choisi pour son efficacité en terme de consommation et de puissance de calcul [30]. La soustraction est réalisée par l'utilisation de l'arithmétique en complément à 2. Enfin, les traitements SWP utilisent la décomposition de l'arbre de *Sklansky* en insérant des multiplexeurs aux endroits adéquats. L'UF travaille sur un résultat de 40 bits afin de permettre des opérations d'accumulation. Des

décaleurs permettent de réaliser des opérations de recadrage sur les entrées, et la sortie. Enfin comme pour le multiplieur, la saturation peut être réalisée dans le même cycle que le calcul.

Afin de réduire l'activité des UF lorsqu'elles ne sont pas utilisées, nous avons inclus des bascules *latch* sur toutes les entrées. Ce surcoût en surface (5% d'augmentation) amène cependant un gain considérable sur la consommation de ces éléments (-23% pour les multiplications 16-bits et -72% pour les multiplications 8-bits).

La table 2-1 résume les performances et le coût des deux unités fonctionnelles pour une technologie STMicroelectronics 0.18 μm. Comme on peut le voir, le délai critique est dû à l'ALU et le pipeline du multiplieur est donc inutile. On remarque cependant que le multiplieur consomme environ deux fois plus que l'additionneur.

	Surface (μm^2)	Délais (ns)	Energie ($10^{-12}J$)
Multiplieur-Additionneur	37 000	3.97	88.90
ALU	28 850	5.33	39.62

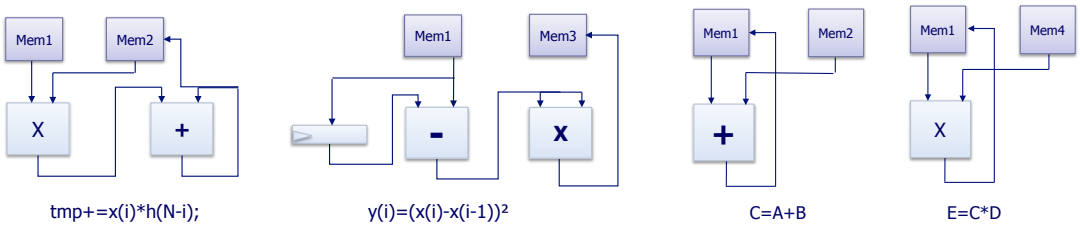
Tableau 2-1 – **Résultats d'implémentation des unités fonctionnelles de DART**. *Ces résultats ont étés obtenus pour une technologie STMicroelectronics 0.18 μm.*

B.2 Paradigme de reconfiguration

La particularité de l'architecture DART est de supporter deux modes de reconfiguration dynamique permettant de s'adapter au contexte d'exécution. En effet, la décomposition des applications en calcul régulier et irrégulier nous a poussé à proposer deux modes de configuration adaptés. Les calculs réguliers, issus des boucles imbriquées, sont les parties consommatrices de temps de calcul et seront optimisés à l'aide de la reconfiguration qualifiée de "hardware". Pour ce type de calcul, une flexibilité totale des DPR est proposée. Ainsi les types d'opérations, les sources des opérandes ainsi que les interconnexions entre tous les composants sont modifiables. Par exemple, sur la figure 2-4a, un DPR est configuré pour deux motifs différents. Cette flexibilité, qui permet de créer n'importe quel motif de calcul dans chaque DPR, a un surcoût en terme de temps de configuration. Cependant, ce surcoût est compensé par les temps de calcul de ces parties de codes. En fonction des caractéristiques de l'application (et de l'utilisation du SCMD) la reconfiguration matérielle nécessite entre 4 et 19 instructions de reconfiguration, chaque instruction étant codée sur 52 bits. Une fois la configuration effectuée, le contrôle est réalisé par l'accès aux données sous le contrôle des AG.

A contrario, les parties irrégulières supportant les aspects contrôle de l'application seront implémentées à l'aide de la reconfiguration dite "software". Dans ce mode, la flexibilité de l'architecture est "dégradée", c'est à dire que l'on ne contrôle pas tous les éléments du DPR. L'exécution se fait selon un motif de type *Read-Modify-Write* (figure 2-4b). Il suffit donc de spécifier l'opération à réaliser ainsi que les sources et destinations des données. Les calculs ne sont plus

optimisés comme pour les cœurs de boucle, mais la reconfiguration s'effectue alors en un seul cycle.



(a) Mode de reconfiguration HW (b) Mode de reconfiguration SW

Figure 2-4 – **Deux modes de reconfiguration supportés par DART**. *Les codes réguliers sont implémentés avec une flexibilité totale des DPR au prix d'un surcoût de reconfiguration. Les codes irréguliers sont supportés en limitant la flexibilité et la performance mais en réalisant une configuration en un seul cycle.*

Grâce à ses deux modes de reconfiguration et au concept de SCMD, DART peut être optimisée pour supporter aussi bien des portions de code de calcul intensif, que de code orienté contrôle. Il est possible en cours d'exécution d'alterner ces différents modes afin de répondre aux besoins d'une application réelle. L'architecture présentée nécessite alors une méthodologie de développement particulière permettant de profiter de ce paradigme de configuration.

B.3 Flot de développement

Afin d'exploiter toutes les capacités de DART, nous avons développé une méthodologie adaptée et complète (figure 2-5). Le point clé de cette méthodologie est d'identifier et de traiter les différentes parties du code d'une application. L'objectif est alors de traiter les codes réguliers provenant de boucles comme des reconfigurations *hardware*, tandis que le reste du code sera implémenté sous forme d'instructions de reconfiguration *software*.

Le point d'entrée du flot de conception proposé (figure 2-5) est le langage C. Le front-end est basé sur l'environnement SUIF [53]. Cet environnement permet de transformer une représentation procédurale (C ou C++) en une représentation interne basée sur des graphes de flot de données et de contrôle (CDFG). Des transformations sur cette représentation permettent alors d'optimiser le code (déroulage de boucles, élimination de codes morts, ...). Nous avons inclus des transformations supplémentaires afin d'optimiser le nombre d'accès aux données [32]. Ce front-end permet alors de séparer les cœurs de boucles et le code irrégulier d'une application.

Le mode de reconfiguration *software* est très proche du mode de fonctionnement d'un processeur VLIW classique. De fait, nous avons utilisé dans le flot de conception le compilateur reciblable CALIFE [19] permettant de générer les mots de configuration adéquats. Nous avons dans un premier temps décrit l'architecture DART en langage ARMOR. Cette description ne décrit l'architecture que dans son mode de fonctionnement dégradé. Nous avons alors implémenté des phases classiques de sélection de codes, d'allocation et d'ordonnancement adaptées

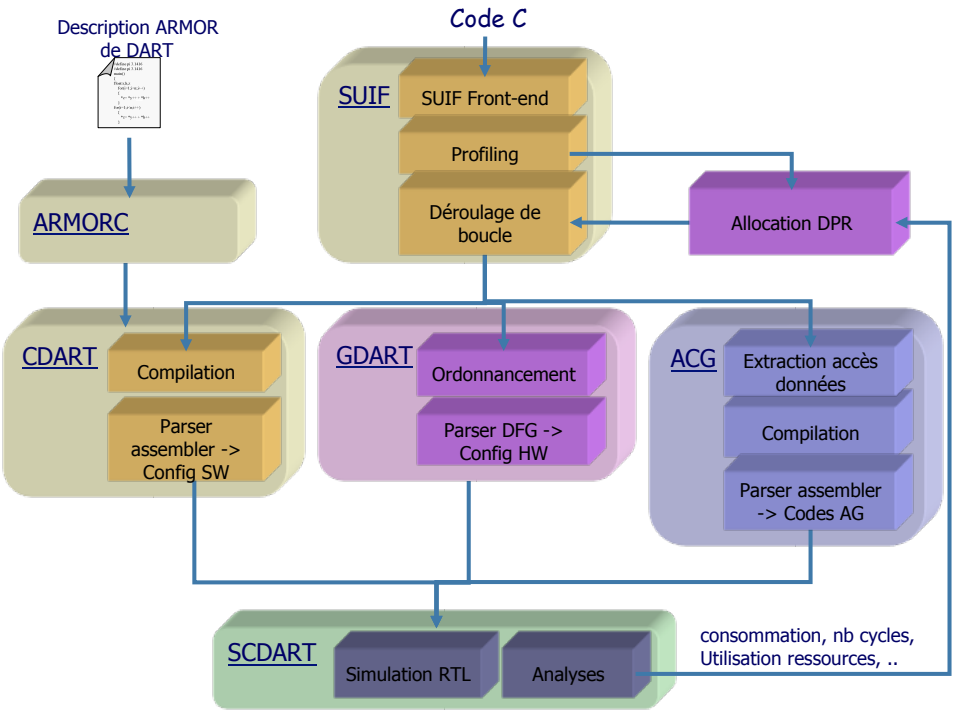


Figure 2-5 – **Méthodologie de développement de DART**. *Après un changement de représentation et des optimisations dans le front-end les configurations software et le code des générateurs d'adresses sont générés à l'aide d'un compilateur reciblable. Les reconfigurations hardware elles, sont construites en utilisant des approches de la synthèse de haut niveau. La méthodologie est complétée par un simulateur SystemC de DART.*

à l'architecture. Ces différentes phases constituent le compilateur CDART (figure 2-5).

Les configurations *hardware* reviennent à créer un chemin de calcul adapté à un motif particulier, ce qui est exactement l'objectif des outils de synthèse de haut-niveau. Le point d'entrée de l'outil gDART (figure 2-5) est le cœur des boucles à optimiser sur le cluster DART. Nous nous sommes basés sur l'expérience de l'équipe en utilisant et en adaptant l'outil GAUT [79].

Finalement, la liaison entre les différentes phases de reconfiguration et la synchronisation globale de l'architecture se fait par l'intermédiaire des générateurs d'adresses. Il reste donc à générer le code des AG, ce qui est le rôle de l'outil ACG (figure 2-5). Comme les AG sont des petits processeurs de type RISC, nous avons réutilisé le compilateur reciblable CALIFE pour cette génération. Cette fois ci les instructions obtenues sont les instructions de manipulations des données tout au long de l'exécution de l'application.

L'ensemble des configurations obtenues peut être validé par un simulateur SystemC de DART (scDART sur la figure 2-5). La modélisation de l'architecture est effectuée au niveau transferts de registres afin d'obtenir des résultats de simulation suffisamment précis en terme de temps de calcul mais aussi de consommation. Cette estimation de consommation a été obtenue grâce à la caractérisation bas niveau de tous les composants de DART, avec l'outil *Design Power* de Synopsys.

B.4 Implémentation et discussion sur la répartition d'énergie

Afin de valider l'architecture et son flot de développement, nous avons implémenté un récepteur WCDMA (Wideband Code Division Multiple Access) complet [63]. Très utilisé dans les télécommunications mobiles de troisième génération, ce récepteur requiert $\approx 4,1$ GOPS. La puissance de calcul effective d'un cluster DART est de 6,2 GOPS sur des données 8 bits. Ceci est rendu possible par la flexibilité du réseau d'interconnexion qui offre un très large éventail de gestion du parallélisme. La reconfiguration dynamique de DART permet alors de supporter l'ensemble des tâches du récepteur WCDMA. Sur l'ensemble de l'application, ces phases de reconfiguration représentent 0,05% du temps d'exécution global.

La puissance consommée par DART pour cette application est de 115mW. Ce résultat a été obtenu pour une implémentation de DART en technologie CMOS 0.18 μm, avec un $V_{DD} = 1.9V$ et une fréquence de fonctionnement de 130 MHz. Selon ce résultat, l'efficacité énergétique de DART est alors de 38.8 MOPS/mW. Nous avons implémenté la même application sur un FPGA Xilinx VIRTEX II. Selon les implémentations réalisées nous avons obtenue entre 0,4 MOPS/mW et 7,7 MOPS/mW au mieux. De même, l'implémentation d'une sous partie de l'application sur un DSP VLIW TMS320C62 montre une EE de 0,3MOPS/mW. Il est important de noter que 79% de la consommation de DART est due aux unités fonctionnelles. La diminution des bitstreams ainsi que les différentes stratégies d'optimisation de la reconfiguration permettent de ne consommer que 0,9 mW pour cette partie, ce qui représente 0,8% de la consommation globale. Le reste de la consommation (≈20%) est dû aux accès mémoire. Ce résultat est rendu possible grâce à l'optimisation de ces accès (utilisation des registres locaux, *broadcast*, ...).

B.5 Intégration VLSI et résultats

L'intégration de DART a été réalisée dans le contexte du projet européen 4MORE qui visait à proposer une plate-forme pour les applications de télécommunication de quatrième génération. L'architecture Fresh [51] est un SoC construit autour d'un réseau asynchrone. Cette architecture contient 23 modules différents, pour une complexité totale de 8 millions de portes (mémoire incluse). Le circuit a été réalisé en utilisant une technologie STMicroelectronics CMOS 0.13 μm. Plusieurs modules de différents partenaires ont été intégrés dans Fresh :

- Un cœur de processeur ARM946ES qui est inclus dans un sous-système avec un bus AMBA.
- Deux IP de calcul intensif, un encodeur turbo de France Telecom R&D et un décodeur convolutionnel (Viterbi) de Mitsubishi/ITE.
- Différentes IP pour communication OFDM conçues par le CEA LETI.
- Un controleur reconfigurable (RAC) conçu par le CEA List.
- Un cluster DART.

La figure 2-6 montre la photo du circuit Fresh ainsi que le "floorplan" des différents modules.

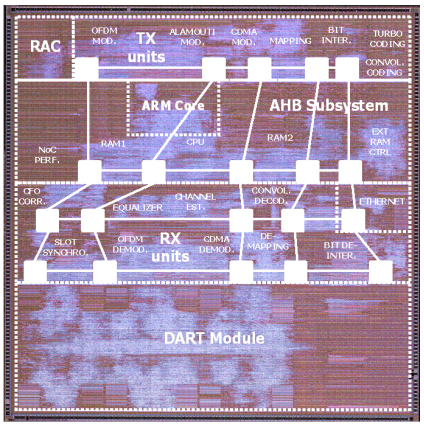


Figure 2-6 – ***Photo du circuit Fresh,*** *incluant le module DART.*

Technologie	0.13 μm CMOS de STmicrolectronics
Tension d'alimentation	1.2 V
Taille du circuit	2.68 mm × 8.3 mm
Fréquence	200 MHz
Consommation moyenne	709 mW
Nbr de portes	2.4 Million
Performances	4800 MOPS sur 32-bit data 9600 MOPS sur 16-bit data

Figure 2-7 – ***Caractéristiques du module DART implémenté.***

Dans le circuit Fresh, DART est prévue pour réaliser l'estimation de canal d'une réception OFDM. Afin de supporter cette application nous avons implémenté deux diviseurs dans la zone permettant d'ajouter de la logique dédiée (figure 2-1). La hiérarchie mémoire a été adaptée afin de répondre au protocole de communication du réseau utilisé. Ainsi deux FIFO ont été ajoutées et toutes les mémoires locales sont à double accès afin de réaliser une lecture et une écriture dans le même cycle. Afin de répondre aux exigences de précision de l'algorithme, les DPR ont été modifiés pour supporter des données codées sur 32 bits. Cette modification a un impact fort sur la consommation mais est nécessaire pour cette application.

Le résultat est un module d'une complexité de 2,4M de portes (y compris la hiérarchie mémoire). La table 2-7 résume les principales caractéristiques du module DART obtenu. Ces résultats notamment en terme d'EE montrent l'efficacité de notre architecture.

C La plate-forme MOZAÏC

Contexte

Cette partie présente les travaux de thèse de Julien Lallet [50] sur la période 2005-2008. Ce travail a pris place dans le cadre du projet CoMap en collaboration avec l'université de Erlangen. Ces travaux visent à la définition d'une plate-forme générique de modélisation et de conception de systèmes embarqués reconfigurables dynamiquement.

Dans le domaine de la conception d'architectures reconfigurables dynamiquement, la modification de certains paramètres influe directement sur trois critères fondamentaux :

1. la flexibilité, qui représente la capacité d'une architecture à s'adapter aux nouvelles applications,
2. la dynamicité, qui elle, caractérise la rapidité à laquelle une architecture pourra s'adapter à de nouvelles applications,

3. et la performance, qui est un critère plus classique de la conception.

La conception d'un système embarqué reconfigurable fait l'objet d'un compromis entre ces trois critères. Ainsi, concevoir une architecture gros grain telle que DART permet d'augmenter les performances et la dynamicité de l'architecture au prix d'une perte de flexibilité. Selon l'application, ou le domaine d'applications visé, il sera alors nécessaire de procéder à des choix d'implémentation qui devront se justifier par rapport aux performances (surface, consommation, fréquence) de l'architecture désirée. Ces contraintes ont amené à une complexification des architectures, de leurs mécanismes de reconfiguration et de leur conception. De manière à répondre efficacement à ce problème, des plates-formes de développement ont été conçues et permettent ainsi d'automatiser certains processus constituant la chaîne de conception d'une architecture. Cela est rendu possible par l'intermédiaire de langages de description haut niveau (ADL – Architecture Description Language) [29]. Cela permet, par une spécification rapide de certains paramètres matériels, de procéder rapidement à la génération d'une architecture et de ses outils de développement [48] (tels que des outils de simulation, de compilation ou encore de synthèse).

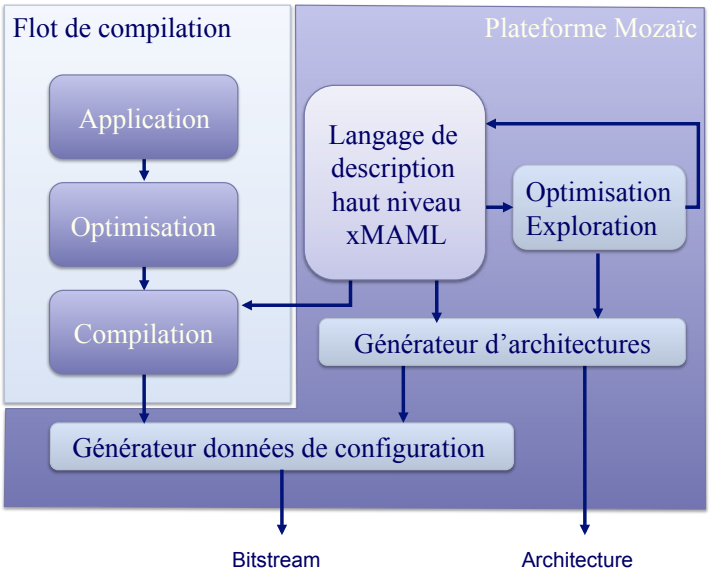


Figure 2-8 – **Plate-forme de développement Mozaïc et son environnement**. *La conception d'un système reconfigurable dynamiquement à l'aide de* Mozaïc *commence par la description haut niveau de l'architecture. L'exploration terminée, la description haut niveau est analysée et conduit à la production automatique du code synthétisable de l'architecture finale. Après la génération de l'architecture, les outils de synthèse pour l'architecture sont paramétrés afin de produire les bitstreams adéquats.*

La plate-forme Mozaïc que nous avons développée (figure 2-8) permet la conception d'architectures reconfigurables dynamiquement par l'introduction automatique de ressources matérielles dédiées et adaptées. Nous avons défini le langage xMAML basé sur l'ADL MAML [47], développé à l'université d'Erlangen. Nous avons alors ajouté les paramètres de spécification nécessaires à la mise en œuvre de la reconfiguration dynamique, ainsi qu'à la spécification d'ar-

chitectures hétérogènes. La plate-forme Mozaïc permet alors de paramétrer les outils requis par l'architecture reconfigurable.

C.1 Le langage de description xMAML

le langage xMAML est utilisé dans Mozaïc afin de fournir les informations utiles à la mise en œuvre du modèle de reconfiguration dynamique. Une description xMAML représente un ensemble de ressources dont le nombre et l'organisation varient en fonction de l'architecture à décrire. La figure 2-9 résume ces différentes ressources. Tout d'abord, les DyRIBox, sont les unités d'interconnexion utiles aux communications internes à l'architecture. Les unités de traitement (PE – Processing Element) sont spécifiées selon leur emplacement dans l'architecture et selon leurs ports de communication nécessaires aux traitements des données, et à la reconfiguration dynamique. La spécification d'un domaine (DBDomain) permet de définir les unités de traitement et d'interconnexions qui font partie de la même zone de reconfiguration.

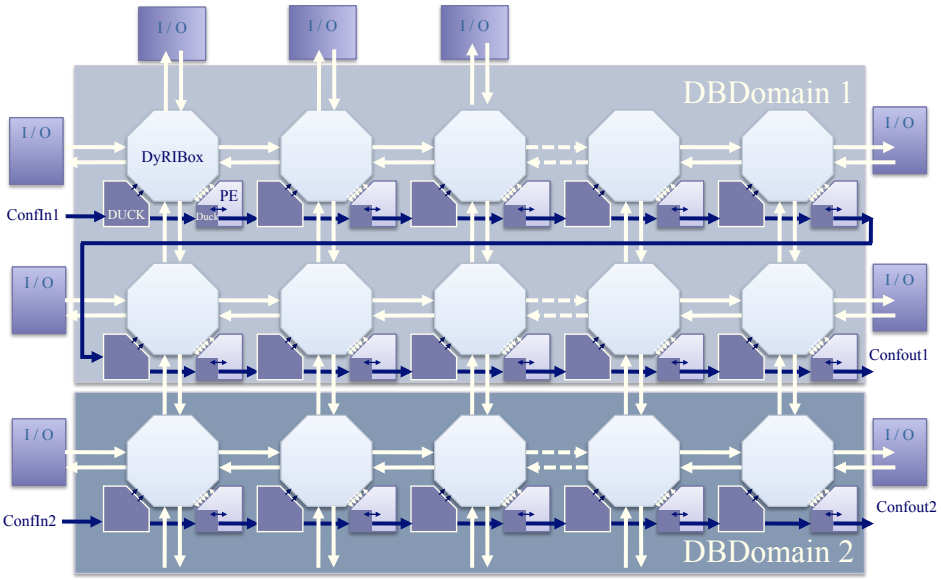


Figure 2-9 – **Paramètres nécessaires à la description d'une architecture en xMAML**. *Une architecture est un ensemble d'unités d'interconnexion (*DyRIBox*), de domaines de reconfiguration (*DBDomain*), d'unités de traitement (PE) et d'un ensemble de ressources de reconfiguration (DUCK). Dans cet exemple, une topologie de type mesh est implémentée.*

Hérité du langage MAML, le PE représente l'unité de calcul de l'architecture. Dans notre cas, c'est la brique de base de la reconfiguration. Ainsi, si l'on décrit une architecture de grain fin, de type FPGA, le PE sera l'équivalent d'un CLB (Configurable Logic Block), alors que si l'on décrit une architecture gros grain, type DART, le PE pourra être un DPR. La plate-forme Mozaïc supporte des descriptions intégrant différents types de PE, ce qui autorise la description d'architectures hétérogènes. Ceci est obligatoire pour la description de FPGA modernes, dans lesquels on retrouve un ensemble de CLB mais aussi des blocs mémoires ou encore de pe-

tits cœurs de calcul dédiés (multiplieur ou unité MAC). L'unité d'interconnexion, *DyRIBox*, est chargée d'assurer la continuité des communications entre les unités de traitement hétérogènes implémentées sur l'architecture, même pendant la reconfiguration. Les différentes ressources doivent être capables de communiquer par l'intermédiaire d'un réseau d'interconnexion flexible et reconfigurable. De plus, l'intégration et l'implémentation de ressources différentes peut nécessiter la mise en place de ressources d'interface chargées de la régularisation des processus de reconfiguration.

L'un des enjeux de la reconfiguration dynamique se situe dans la possibilité de procéder à la reconfiguration partielle d'une architecture. Il est alors nécessaire, dès la description de l'architecture, de définir des zones de reconfiguration indépendantes les unes des autres. Nous avons étendu la notion de domaines du langage MAML afin de regrouper les ressources appartenant à une même zone de reconfiguration (*DBDomain*). Un domaine est donc une zone où la reconfiguration dynamique est exécutée de manière homogène en termes de taille de mots de configuration, et de temps nécessaire à la reconfiguration. Cela permet de pouvoir procéder à des reconfigurations dynamiques partielles indépendantes et donc parallèles. Cette configuration prend place grâce aux DUCK.

L'objectif de Mozaïc est de permettre la conception d'architectures reconfigurables dynamiquement hétérogènes par la virtualisation des caractéristiques des ressources de traitement utilisées. Pour cela, nous avons développé un modèle de reconfiguration paramétrable et suffisamment flexible, pour permettre la conception de tous types d'architectures. La reconfiguration dynamique par multi-contexte permet une accélération des processus de reconfiguration significative, au prix de l'augmentation des ressources de mémorisation. Cela contribue à une inefficacité énergétique incompatible avec les contraintes actuelles. La méthode que nous proposons repose sur l'utilisation d'une seule mémoire de contexte parallèlement aux ressources de configuration. Cela implique l'utilisation de solutions architecturales afin de pouvoir maintenir les contraintes temporelles et la flexibilité requises par les applications actuelles. Nous avons pour ce faire séparé les chemins de données de configuration et les ressources de configuration à proprement parler (figure 2-9).

Ceci se traduit par l'introduction de nouvelles unités baptisées DUCK (**D**ynamic **U**nifier and re**C**onfiguration bloc**K**), qui permettent de faire l'interface entre le chemin de reconfiguration et l'unité à reconfigurer correspondante. Un DUCK est une mémoire de configuration locale par laquelle passe le bus de configuration. C'est une mémoire supplémentaire connectée en parallèle, qui contient la configuration future adaptée à l'unité avec laquelle elle est connectée. Le DUCK est donc dédié à une unité de traitement (PE), à une unité d'interconnexion (*DyRIBox*) ou à un bloc d'entrée/sortie. Les unités de reconfiguration sont reliées entre elles, sous forme d'une chaîne, de manière à pouvoir propager les configurations. Selon le nombre d'unités de traitement, d'interconnexion et d'entrées/sorties, le nombre d'unités de reconfiguration augmente, de sorte que la propagation des configurations n'est plus suffisamment rapide pour maintenir les contraintes temporelles imposées par les applications. La création des *DBDomain* permet

de remédier à cela. Ces zones de reconfiguration sont contrôlées individuellement mais peuvent communiquer entre elles. Un domaine est donc constitué d'un ensemble d'unités de reconfiguration formant une seule chaîne de propagation. Hormis les PE, tous les autres éléments de l'architecture sont générés ou paramétrés par Mozaïc et ont un impact sur la création du DUCK concerné :

Les unités d'interconnexion (DyRIBox) doivent accepter tous types de connexion. Par conséquent, celles-ci sont paramétrables en nombres de ports d'entrée, en nombre de ports de sortie ainsi qu'en nombre de ports bidirectionnels. De plus, selon les applications devant être implémentées, la taille des données qui seront transmises peut varier. Il doit être également possible de modifier un schéma de connexion. Les unités d'interconnexion doivent donc être reconfigurables dynamiquement. Cela implique de réduire le nombre des données de reconfiguration. Les unités d'interconnexion peuvent donc accepter des restrictions dans le schéma de connexions supporté, en interdisant à une entrée de se connecter sur une sortie par exemple. Enfin, les DyRIBox permettent la conception de topologies d'interconnections diverses (mesh sur la figure 2-9).

Les unités d'entrées/sorties (I/O) ont pour rôle de permettre les communications des PE avec le monde extérieur. La nature de ces communications va déterminer le nombre, la taille et la nature des ports de communication. Afin de prévenir les problèmes de compatibilité, des mémoires tampons (paramétrables en taille) sont présentes dans ces blocs et permettent des adaptations de protocoles.

Les unités de reconfiguration (DUCK) réalisent l'interface entre les ressources de contrôle de la reconfiguration produites par Mozaïc et les PE implémentés au sein d'un même domaine. Pour cela, les DUCK doivent stocker les configurations localement puis les transférer vers le PE concerné selon un protocole prédéfini. Afin de gérer la reconfiguration de manière efficace, il est nécessaire de connaître le nombre de bits et de cycles nécessaires à la reconfiguration, ainsi que le nombre et la taille des ports réservés à cet usage. La spécification du protocole de reconfiguration n'est pas nécessaire pour les unités d'interconnexion ni pour les unités d'entrées/sorties, celles-ci étant elles-mêmes générées par Mozaïc.

Les ressources de contrôle de la reconfiguration d'un domaine de reconfiguration permettent plusieurs ajustements de la flexibilité. La génération de chaque DUCK nécessite la spécification de la taille du bus de reconfiguration du domaine. Il est également possible de réunir plusieurs domaines de reconfiguration de manière à ce qu'ils soient gérés par un seul contrôleur. Afin d'augmenter l'efficacité de la reconfiguration, la préemption et la gestion des priorités d'implantation des tâches (configurations ou contextes) ont été mises en œuvre. La gestion de la préemption implique la spécification de paramètres tels que la taille de la mémoire tampon qui servira à temporiser l'ordonnancement des différentes interruptions, les différents niveaux d'interruptions ainsi que le nombre d'interruptions par niveau.

C.2 Virtualisation de la reconfiguration dynamique : le concept DUCK

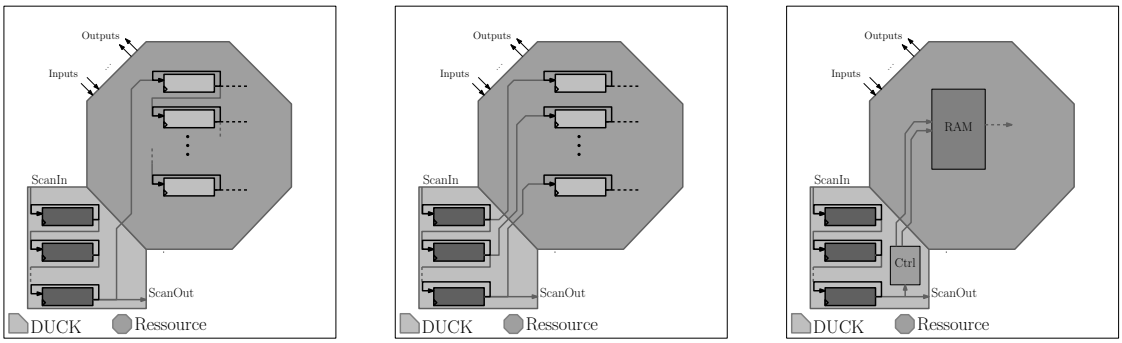
Le DUCK est donc l'interface permettant l'adaptation des différents éléments de l'architecture dans l'infrastructure Mozaïc. Ce concept a été élaboré afin de répondre à trois problématiques majeures dans le développement d'architectures reconfigurables dynamiquement :

1. l'accélération des processus de reconfiguration qui permet une réelle utilisation de la reconfiguration dynamique dans des applications complexes.
2. La gestion des mécanismes de préemption. Celle-ci autorise des techniques de gestion avancées pour les architectures qui doivent supporter plusieurs applications ou répondre à des sollicitations externes.
3. La gestion homogène de la reconfiguration dynamique quelles que soient les ressources reconfigurables implémentées, et ce afin de pouvoir concevoir différentes architectures hétérogènes adaptées à l'application.

S'appuyant sur une mémoire multi-contexte, l'utilisation des DUCK permet une exécution parallèle des processus de reconfiguration et des processus de traitement. Autrement dit, l'acheminement d'une configuration se fait pendant les phases de calculs sans avoir à arrêter le traitement en cours. Pour cela, le concept DUCK intègre des chaînes de registres (figure 2-9), sur le principe des registres *scanpath* utilisés dans les techniques de test. Dans ce type de système, on peut configurer les connexions de sorte que l'ensemble des registres d'un même domaine constitue un seul et unique registre à décalage. Afin d'adapter cette technique à la reconfiguration dynamique, nous avons introduit des mécanismes permettant l'échange entre le contexte "futur" contenu dans le DUCK et le contexte "actuel" à modifier dans les ressources de l'architecture. La propagation des reconfigurations (ConfIn sur la figure 2-9) permet alors, grâce à la chaîne de *scanpath*, de supprimer les données d'adressage de la mémoire de configuration. Suite à l'échange entre les contextes, les mêmes ressources sont utilisées pour récupérer le contexte de la configuration passée (ConfOut, figure 2-9). Cette extraction rend possible la sauvegarde de contexte et donc la préemption de tâches sans surcoût matériel et sans pénaliser l'exécution de l'application.

Dans le domaine de la reconfiguration dynamique, différents modes de reconfiguration sont utilisés. Par exemple dans les FPGA du commerce, la reconfiguration se fait en adressant des mémoires de type SRAM, alors que dans le processeur reconfigurable DART, les configurations sont changées par l'intermédiaire de registres de configuration. Nous avons alors développé trois modes de fonctionnement afin de s'adapter à ces différentes méthodes.

Quel que soit le mode choisi, l'acheminement d'une nouvelle configuration se fait en trois étapes. En fonctionnement "courant" les ressources de reconfiguration du DUCK ne sont pas sollicitées, ce qui correspond à une phase d'exécution par les ressources de l'architecture. La première étape d'une phase de reconfiguration, consiste alors à acheminer un nouveau bitstream dans le DUCK. Pour cela on utilise les registres *scanpath*, de sorte que les bits de configuration sont



(a) DUCK en mode séquentiel (b) DUCK en mode parallèle (c) DUCK en mode adressage

Figure 2-10 – **Différents modes de reconfiguration supportés par le DUCK**. *La configuration d'une ressource se fait toujours selon trois phases. La première phase permet la propagation d'un nouveau contexte en utilisant les registres scanpath des DUCK. La deuxième phase correspond à la reconfiguration de la ressource connectée au DUCK en utilisant une méthode adaptée. Enfin la dernière phase correspond à l'extraction du contexte précédent, la configuration du DUCK est identique à la phase de propagation.*

acheminés cycle par cycle. Ce processus reste le même pour extraire la configuration suite à l'échange des contextes dans le cadre de la préemption. La deuxième étape de la reconfiguration permet l'échange de contexte entre le DUCK et la ressource associée (*switch* de contexte) et réalise la reconfiguration des ressources à proprement parler. C'est cette phase qui supporte les différents modes de reconfiguration afin de s'adapter à la ressource, et éventuellement de faire des compromis temps de reconfiguration/surface. Le premier mode envisagé est qualifié de "séquentiel" (figure 2-10a). Dans ce mode, les registres de configuration de la ressource sont interconnectés, eux aussi, sous la forme d'un registre à décalage. Ainsi la reconfiguration nécessite plusieurs cycles d'horloge mais réduit les ressources de routage nécessaires. Dans le mode de reconfiguration parallèle (figure 2-10b), chaque registre de configuration de la ressource est directement connecté à son registre DUCK, permettant une reconfiguration en un cycle. Le dernier mode supporte la reconfiguration par "adressage", utilisé par exemple dans les FPGA dont la mémoire de configuration est une SRAM. Le DUCK doit cette fois pouvoir adresser cette mémoire et aller y écrire la configuration stockée dans ses registres (figure 2-10c).

Nous avons mis en œuvre la plate-forme Mozaïc par la génération des ressources nécessaires à la reconfiguration dynamique sur différentes architectures et notamment l'architecture DART. Je ne présente dans la suite que les résultats concernant la conception d'un FPGA enfoui (eFPGA).

C.3 FPGA enfoui (eFPGA)

Nos expérimentations sur la génération d'un FPGA embarqué basé sur une matrice de cellules logiques comparables à la famille XC4000 de chez Xilinx nous ont amené à étudier cette architecture plus avant.

C.3-1 Architecture

Nous avons dans un premier temps décrit la structure souhaitée en xMAML comme le présente le listing 2-1. Ce FPGA 4x4 (pour l'exemple dans ce document) est basé sur des DyRIBox pour l'interconnexion (ligne 3). Ces dernières comportent quatre entrées et quatre sorties de un bit chacune. L'élément de calcul est un CLB (ligne 2) comparable à ceux du XC4000 de Xilinx (aire Ⓑ, Ⓒ et Ⓓ de la figure 2-11). Il comporte 4 entrées et deux sorties (ligne 10 et 11). Tous ces éléments sont inclus dans un seul domaine avec une interconnexion de type mesh (ligne 15). Les lignes 20 à 27 permettent de décrire sous une forme condensée l'ensemble des connexions du domaine.

```
1  <ProcessorArray name="fpga" version="1.0">
2     <PElements name="clbXC4000" rows="1" columns="1"/>
3     <PEInterconnectDyRIBox name="DBox">
4        <ReconfigurationTime cycle="1"/>
5        <DBPorts>
6           <Inputs number="4" bitwidth="1" />
7           <Outputs number="4" bitwidth="1" />
8        </DBPorts>
9        <PElementsPorts>
10          <Inputs number="4" bitwidth="1"/>
11          <Outputs number="2" bitwidth="1" />
12       </PElementsPorts>
13    </PEInterconnectDyRIBox>
14    <DBDomain name="fpga" MemoryScan="enabled">
15       <Interconnect type="mesh" >
16          <PEInterconnectDyRIBoxBase name= "DBox"/>
17          <PEMeshBase name= "clb"/>
18       </Interconnect>
19       <ElementsPolytopeRange>
20          <MatrixA row = " -1 0"/>
21          <MatrixA row = " 1 0"/>
22          <MatrixA row = " 0 -1 "/>
23          <MatrixA row = " 0 1 "/>
24          <VectorB value = " 0"/>
25          <VectorB value = " 4"/>
26          <VectorB value = " 0 "/>
27          <VectorB value = " 4"/>
28       </ElementsPolytopeRange>
29    </DBDomain>
30  </ProcessorArray>
```

Listing 2-1 – ***Description xMAML d'un FPGA.*** *Ce FPGA est constitué d'un seul domaine, avec une topologie mesh. L'interconnexion est réalisée grâce à des DyRIBox, les PE sont des CLB.*

La génération de la logique supportant la reconfiguration dynamique donne l'architecture de la figure 2-11 pour un DUCK en mode séquentiel. Le CLB implémente ses registres de configuration (zone Ⓐ de la figure 2-11) permettant de le configurer en mode séquentiel ou combinatoire ou encore de choisir l'entrée utilisée pour le calcul de la retenue. Le réseau de multiplexeurs

(zone Ⓑ) permet l'utilisation des ressources de la LUT (Look Up Table) en mode RAM. Les ressources nécessaires au calcul de la retenue pour les opérations arithmétique sont représentées zone Ⓒ. Enfin, la LUT utile à l'implémentation de la fonction logique est représentée zone Ⓓ avec sa bascule de sortie.

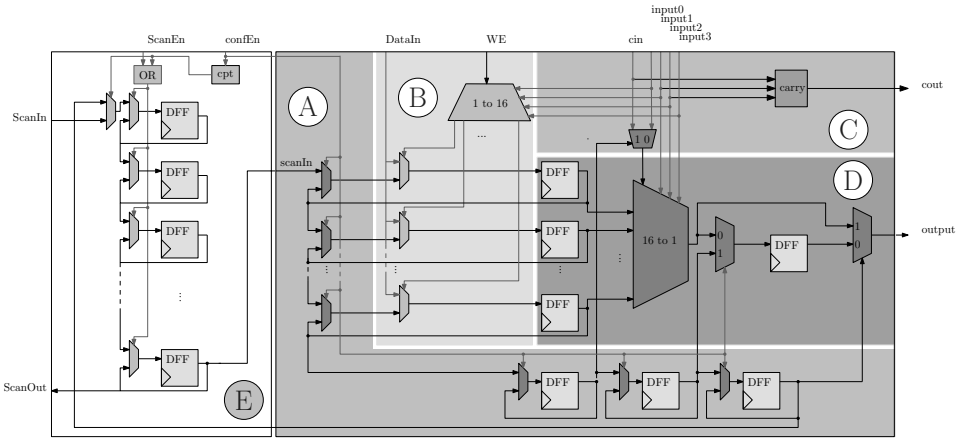


Figure 2-11 – **DUCK séquentiel généré pour le eFPGA**. *Le DUCK représenté zone Ⓔ, est connecté à un bloc logique de FPGA. La zone Ⓐ regroupe les registres de configuration, la zone Ⓑ contient les ressources nécessaires à l'utilisation du bloc logique en RAM, la zone Ⓒ contient les ressources de calcul de retenue et la zone Ⓓ la LUT et sa bascule associée.*

Le chemin de configuration du bloc logique passe par l'ensemble des registres de configuration et des registres de LUT. Dans cette architecture, 19 cycles d'horloges sont nécessaires au processus de reconfiguration pour acheminer l'ensemble du contexte depuis le DUCK (aire Ⓔ figure 2-11) jusqu'aux registres de configuration. Ces 19 cycles correspondent aux 19 bits nécessaires à la configuration de la LUT à quatre entrées, et des trois registres de configuration. Ces trois registres permettent de spécifier la pré-configuration du registre de sortie, ainsi que l'utilisation de celui-ci en sortie du bloc logique et enfin la validation de l'entrée *cin* comme entrée de la LUT. Pour un FPGA embarqué composé de $n \times m$ blocs logiques, le temps de propagation de la configuration "future" est de $t_f = n \times m \times 19$ cycles d'horloges (uniquement pour les ressources de calcul). Le processus d'échange de configuration étant exécuté en parallèle pour l'ensemble des DUCK, la reconfiguration physique de chaque CLB se fait alors en $t_r = 19$ cycles d'horloge.

C.3-2 Flot de compilation

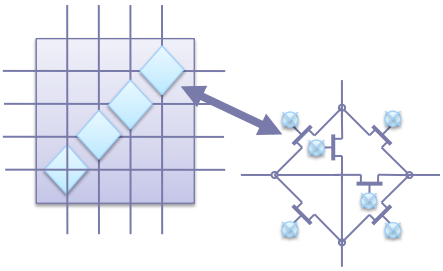
Nous avons aussi construit un flot de conception adapté pour notre eFPGA, en utilisant ABC de l'université de Berkeley [73] pour la synthèse logique à partir de descriptions Verilog d'applications. Le placement routage est, quant à lui, réalisé grâce à l'outil T-VPACK, qui fait parti de la suite logicielle VPR [46] de l'université de Toronto. Pour fonctionner, VPR nécessite une description du FPGA et de ses composants. Cette description est générée automatiquement à partir de la description xMAML du composant (figure 2-8). Les fichiers issus des outils précédents ne sont pas directement exploitables par Mozaïc. Nous avons alors développé les outils

d'interprétation et de conversion des fichiers de placement-routage produits par VPR. Ainsi un générateur de bitstreams adapté a été développé.

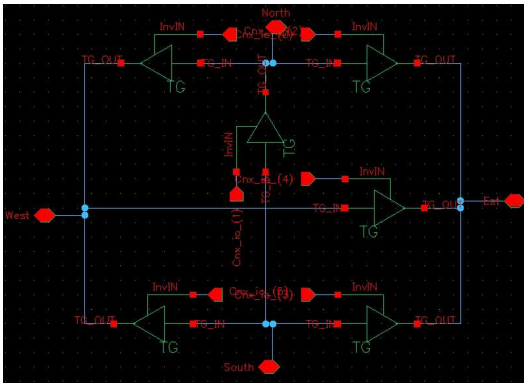
Afin de rendre le eFPGA utilisable et efficace, nous avons commencé à réaliser son implémentation au niveau transistors pour optimiser certaines parties du circuit.

C.3-3 Implémentation VLSI du eFPGA

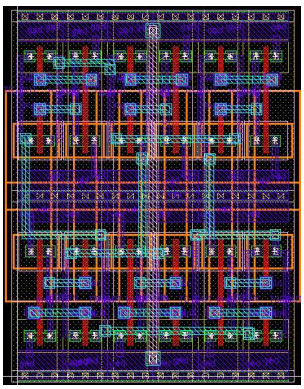
L'efficacité du paradigme de reconfiguration développé dans Mozaïc, ainsi que la volonté d'avoir la maîtrise de l'architecture nous ont encouragés à optimiser le cœur du eFPGA. Ainsi nous avons commencé à travailler sur l'implémentation physique des DyRIBox, et nous travaillons actuellement sur le développement du point mémoire bi-contexte au niveau transistor. Le dessin des masques (*layout*) et le schéma des circuits sont réalisés sous *Cadence Virtuoso*, pour une technologie 0.13 μm de STMicroelectronics.



(a) SwitchBox de type Disjoint



(b) Schéma d'un point d'interconnexion



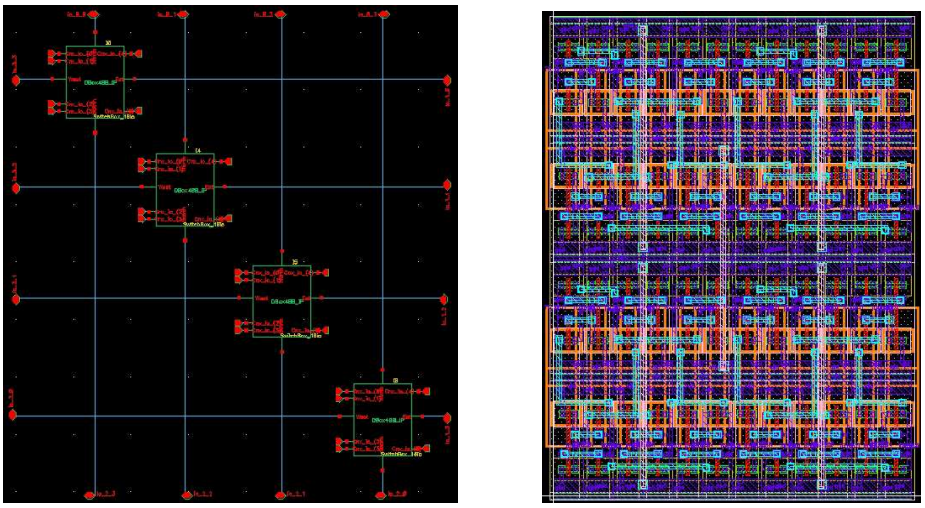
(c) Layout d'un point d'interconnexion

Figure 2-12 – **Implémentation des SwitchBox**. *La SwitchBox est de type disjoint. Elle est construite autour de 4 points d'interconnexion constitué de 6 portes de transfert.*

Nous avons implémenté une SwitchBox de type disjoint présentée sur la figure 2-12a. Ce type d'architecture permet un bon compromis flexibilité-coût d'implémentation. Dans ce type de matrice, il y a 6 transistors (*pass-transistor*) pour chaque point d'interconnexion. Les "pass-transistor" sont commandés par des bits de configuration et travaillent donc comme des interrupteurs programmables. Bien que plus grosses, nous avons implémenté des "transfer-gate", en lieu et place des "pass-transistor", afin de ne pas dégrader les signaux au sein du eFPGA. Le

schéma global d'un point d'interconnexion est alors présenté sur la figure 2-12b. La figure 2-12c montre le layout de ce point d'interconnexion.

Nous trouvons dans la matrice du eFPGA 3 types de DyRIBox :
– des matrices de 8 et 12 entrées/sorties, situées dans les coins et sur les bords du FPGA.
– des matrices complètes de 16 entrées/sorties (figure 2-13a), au milieu du circuit.
Les DyRIBox 8 et 12 entrées-sorties sont des matrices tronquées dans la mesure où elles se trouvent sur les bords du FPGA. La figure 2-13 montre l'implémentation de la matrice complète nécessitant 16 entrées-sorties. Cette DyriBox 16 intègre quatre SwitchBox et nécessite 300 μm^2, alors que les DyriBOX 8 et 12 occupent respectivement 53 μm^2 et 151 μm^2. Ces résultats divisent par ≈200 les surfaces obtenues par synthèse logique à partir de la description xMAML. Les simulations post-layout avec extraction des paramètres montrent que les DyRIBox continuent de fonctionner au delà de 3GHz (jusqu'à une fréquence de 5,5GHz pour la plus petite).



(a) Schéma (b) Layout

Figure 2-13 – **Implémentation d'une DyRIBox de type 16 entrées/sorties**. *Constituée de 4 SwitchBox, la DyRIBox nécessite 300 μm^2.*

Nous travaillons actuellement à l'intégration de la mémoire de configurations du circuit. Cette mémoire supporte de manière efficace l'approche multi-contextes et *switch* de configurations.

D Conclusions et perspectives

Les systèmes sur puce actuels (comme Fresh) comportent plusieurs blocs d'origines diverses intégrés sur un même substrat. Pour composer avec les impératifs de coûts et de performances relatifs au champ d'applications envisagés, le couplage d'un processeur (au sens large) avec des accélérateurs matériels reconfigurables semble être une solution particulièrement attractive, la partie reconfigurable permettant d'implémenter efficacement différents traitements à des ins-

tants disjoints. Il est alors nécessaire d'étudier les mécanismes permettant un échange optimal d'informations entre le processeur et son co-processeur reconfigurable. Nous avons commencé à étudier l'interaction entre un cœur de processeur et un cluster DART. Les travaux que nous menons visent à définir des métriques permettant l'évaluation des performances et des échanges dans l'architecture. Des tests de couplage ont été effectués dans le but d'optimiser les communications au sein de cette plate-forme reconfigurable [APSB04]. Les résultats obtenus ont montré que le couplage architecture gros-grain - processeur n'était pas la meilleure solution.

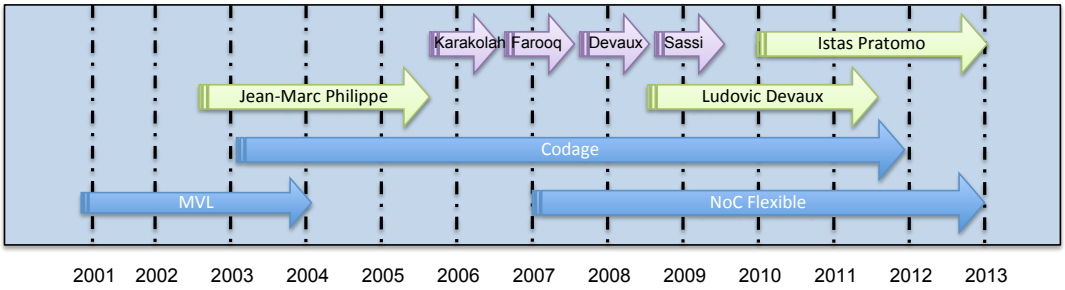
Afin d'aider au développement d'architectures reconfigurables dynamiquement, nous avons développé un langage de description d'architectures à haut niveau, xMAML. Ce langage permet l'intégration et la gestion cohérente d'unités de traitement hétérogènes et génère automatiquement les ressources adaptées qui seront nécessaires à la mise en œuvre de la reconfiguration dynamique. Une analyse de la description xMAML permet une première estimation rapide de la surface de silicium et de la consommation d'énergie induite par la mise en œuvre de cette reconfiguration dynamique. De plus, l'analyse de la description permet également de déterminer la taille des flots de données de configuration propre aux ressources de connexions et aux unités de traitement ainsi qu'à l'estimation du nombre de domaines de reconfiguration nécessaires selon l'application choisie. Une des problématiques vient de l'intégration de Mozaïc avec les plates-formes de développement tels que VPR et ABC. En effet, nous avons dû développer des outils spécifiques, notamment pour l'analyse et la génération automatique des données de configuration sur FPGA. Nous travaillons donc à la définition de passerelles permettant d'automatiser le développement de méthodologie de conception. Basé sur l'environnement Biniou [49], le principe vise à générer un langage de description intermédiaire permettant de construire une chaîne de compilation virtuelle.

La dernière perspectives de ces travaux vise à développer les points mémoires auto-reconfigurable requis par le eFPGA. Cette mémoire est utilisée pour sauvegarder deux bitstreams différents. La particularité vient du fait de pouvoir échanger les informations entre les deux contextes afin de supporter la préemption de tâches au sein du eFPGA. Nous travaillons actuellement sur une technologie CMOS classique, mais l'intégration de points mémoires magnétiques [37] ouvre de nouvelles perspectives à ce travail. En effet, les nouvelles technologies *Thermally Assisted Switching* (TAS)-MRAM, ne requièrent que peu de courant pour être programmées. Ces mémoires présentent l'avantage de n'avoir qu'un très faible surcoût pour l'ajout de contextes supplémentaires. Dés lors que la logique de transformation électrique est déjà implémentée, il est alors possible d'intégrer à faible coût plus de deux contextes.

Chapitre

3 Interconnexion flexible et efficace

Résumé : Les technologies sub-microniques amènent de nouvelles contraintes de conception nécessitant d'appréhender des phénomènes physiques jusqu'alors négligeables et négligés. Le deuxième effet de l'avènement de ces technologies vient de la conception de RSoC dynamique, qui nécessitent un support de communication efficace et flexible. Je présente dans ce chapitre mes travaux autour de la problématique de l'interconnexion dans les SoC modernes. Les personnes impliquées ainsi que les projets supports de ces travaux, sont présentés sur la figure suivante.



Sommaire

A Introduction

L'avènement des technologies sub-microniques (DSM – Deep Sub-Micron) a eu deux impacts sur la conception de l'interconnexion : (1) La diminution des tailles de gravure des circuits et l'augmentation des niveaux de métallisations qui entraînent des phénomènes physiques jusqu'alors négligeables ; (2) la possibilité d'implémenter sur une même puce des architectures parallèles, hétérogènes et dynamiques, qui nécessitent un support de communication efficace et flexible. De ce fait, de nouvelles contraintes apparaissent sur l'interconnexion. La première est que le compromis classique bande passante - latence n'est plus forcément fixe sur l'ensemble du circuit. On perçoit ainsi le fait que l'interconnexion entre différents domaines de communication devra être flexible pour s'adapter aux différents besoins des applications sur la puce [33]. A l'extrême, l'approche System-In-Package (SiP) met en commun des ressources analogiques et numériques ou synchrones et asynchrones. L'interconnexion doit alors créer un pont entre ces différents domaines. Le "passage à l'échelle" (*scalabilité*) est un autre challenge puisqu'il faut interconnecter de plus en plus de modules entre eux. La principale difficulté ici n'est pas forcément d'accroître le nombre de modules sur l'interconnexion mais de ne pas trop dégrader les performances en bande passante et surtout en latence.

La diminution de la taille des transistors a pour conséquence l'augmentation du temps de traversée des fils, et induit une modification des propriétés physiques des matériaux mis en oeuvre, ainsi que l'apparition ou l'amplification de phénomènes physiques rendant la propagation des signaux non sûre [35]. Ainsi, les fils deviennent les principales sources de délai dans le système, et introduisent aussi des erreurs pendant la transmission des données. De plus, dans le cadre de la conception de systèmes embarqués, la contrainte de consommation d'énergie rajoute une difficulté majeure sur la conception de cette interconnexion.

Cette question de l'interconnexion est vraiment primordiale puisque l'on en arrive maintenant à parler de conceptions centrées communications [60] (Communication-Centric). En effet, le réseau d'interconnexion doit avoir des propriétés très étendues, être flexible et très performant. Le goulet d'étranglement dans les futurs systèmes proviendra à coup sûr non pas des ressources de calcul, mais du transit d'informations entre elles. Une réponse viable et de plus en plus envisagée est alors la création de réseaux sur puce (NoC – Network on Chip) [10].

J'ai étudié dans mes travaux l'ensemble des couches de la conception d'interconnexions flexibles, faibles consommation et sûres en terme de transmission. Ainsi, nous avons proposé au niveau physique et liaison, différentes techniques basées sur l'utilisation d'une logique utilisant un nombre d'états logiques supérieurs à deux (MVL – Multiple Value Logic) [64]. Nous avons aussi proposé des codes correcteurs d'erreurs qui augmentent la fiabilité de la transmission. La reconfiguration dynamique permet d'allouer en-ligne des tâches en fonction des besoins de l'application et/ou de l'environnement. L'interconnexion doit alors supporter la flexibilité requise par ce type d'approche. Nous avons donc conçu un NoC adapté à cette problématique.

B Couches physique et liaison

Contexte
Cette partie porte sur des travaux qui ont commencés dans le cadre de la thèse de Jean-Marc Philippe [70] sur la période 2001-2003, et que je continu en collaboration avec Stanislaw Piestrack de l'université de Metz. Ces travaux visent à réduire la consommation des liaisons, tout en optimisant leur fiabilité.

Dans les technologies CMOS modernes, les interconnexions représentent une part non négligeable de la consommation énergétique (qui peut aller jusqu'à 50% [92]) et de la surface de la puce. Des contraintes (telles que la surface ou la vitesse) sur les SoC imposent de recourir à de nouvelles interconnexions haute vitesse et basse consommation. Parmi l'ensemble des problèmes introduits par les technologies DSM (IRDrop, électromigration, soft errors, Ldi/dt drop), le *crosstalk* (interférences causées par le couplage qui existe entre les fils adjacents d'un bus) pose actuellement le plus de problèmes aux concepteurs [35].

Le phénomène de *crosstalk* est directement influencé par les capacités de couplage (C_c) entre le fil victime V et ses deux agresseurs ($A1$ et $A2$). Ces capacités de couplage dépendent de constantes technologiques mais aussi des dimensions des fils (comme leur longueur, leur épaisseur et l'espacement existant entre eux).

On peut résumer les différents effets liés au phénomène de *crosstalk* en trois catégories [80]. Le premier problème réside dans l'augmentation du délai de propagation des informations sur les fils du bus. Le *crosstalk* introduit un facteur de délai (g), montré dans le tableau 3-1 où $r = C_c/C_s$, avec (C_s) la capacité d'un fil par rapport au substrat. Dans ce tableau, $\uparrow$ représente une transition montante, $\downarrow$ représente une transition descendante et enfin $-$ signifie qu'il n'y a aucune transition sur le fil. Si l'on note R_{fil} la résistance du fil, g est le rapport entre le délai de propagation calculé en tenant compte du *crosstalk* $T_P(C) = g.R_{fil}.C_s$ et le délai calculé sans influence du *crosstalk* $T_P = R_{fil}.C_s$. Ainsi, on obtient la capacité effective C_{eff} en posant $C_{eff} = g.C_s$ et donc $T_P(C) = R_{fil}.C_{eff}$.

C_{eff}	Transitions (A_1, V, A_2)				g
C_S	$(\uparrow,\uparrow,\uparrow)$	$(\downarrow,\downarrow,\downarrow)$			1
$C_S + C_C$	$(-,\uparrow,\uparrow)$	$(-,\downarrow,\downarrow)$	$(\uparrow,\uparrow,-)$	$(\downarrow,\downarrow,-)$	1+r
$C_S + 2.C_C$	$(-,\uparrow,-)$	$(-,\downarrow,-)$			1+2.r
	$(\uparrow,\uparrow,\downarrow)$	$(\uparrow,\downarrow,\downarrow)$	$(\downarrow,\uparrow,\uparrow)$	$(\downarrow,\downarrow,\uparrow)$	
$C_S + 3.C_C$	$(-,\uparrow,\downarrow)$	$(-,\downarrow,\uparrow)$	$(\uparrow,\downarrow,-)$	$(\downarrow,\uparrow,-)$	1+3.r
$C_S + 4.C_C$	$(\uparrow,\downarrow,\uparrow)$	$(\downarrow,\uparrow,\downarrow)$			1+4.r

Tableau 3-1 – **Effets du crosstalk**. *Capacité effective (C_{eff}) et facteur de délai (g) d'un fil victime en fonction des transitions sur les trois fils (A_1, V, A_2).*

Dans le cas le plus favorable, quand les trois fils effectuent une transition dans la même direction,

le délai sur le fil victime correspond au délai sans *crosstalk* (i.e. $g = 1$), comme s'il n'existait aucun couplage capacitif entre les fils. Cependant, dans le cas d'une conception synchrone, l'horloge globale du bus doit être adaptée en regardant seulement le pire cas (i.e. $g = 1 + 4.r$) pour s'assurer de l'intégrité des données transmises. Dans une situation tout à fait plausible dans laquelle $C_c = C_s$, le délai de propagation des données sur un bus peut être multiplié par cinq ou plus [76]. Ainsi, le phénomène de *crosstalk* ralentit considérablement la vitesse des interconnexions et donc la bande passante des NoC.

Le bruit induit par le phénomène de *crosstalk* est un second problème dans la conception de circuits en technologie DSM. La capacité de couplage entre deux fils adjacents introduit un lien permanent entre eux [27]. Une transition sur un fil affecte les deux fils adjacents en leur appliquant un pic de tension. Avec la réduction des dimensions dans les technologies submicroniques, le *crosstalk* a une part relative de plus en plus importante dans le niveau général de bruit d'un circuit car les capacités de couplage entre les fils adjacents augmentent (comparativement à la diminution de la tension d'alimentation).

Enfin, le *crosstalk* contribue à augmenter la consommation d'énergie des circuits du fait de l'augmentation de la capacité effective des fils (C_{eff} dans le tableau 3-1) [52]. Comme la consommation d'une porte ou même d'un système entier dépend en première approximation de sa capacité, le *crosstalk* augmente donc la consommation globale.

Il est donc primordial de réduire les effets du *crosstalk* pour optimiser les performances de l'interconnexion [60]. Les techniques classiques pour y parvenir sont, soit d'éliminer les transitions pire-cas (éliminer les configurations de transmission de type $1 + 3.r$ et $1 + 4.r$), soit de diminuer les capacités de couplage entre fils en augmentant les distances entre fils ou en incluant des fils de blindage (*shielding*). Ces dernières solutions ont un impact fort sur la surface, la consommation et les performances de la liaison.

B.1 Interconnexion multi-valuée

Introduite pour améliorer les performances et diminuer le surcoût d'interconnexion, la logique MVL consiste à coder plus de deux états (typiquement trois ou quatre) sur un seul fil. L'intégration d'un circuit complet en logique MVL est encore actuellement du domaine de la recherche. Par contre, cette logique a des intérêts majeurs en ce qui concerne l'interconnexion. L'utilisation de la MVL permet soit d'augmenter la bande-passante d'une liaison en faisant passer plus d'informations en un seul cycle [68], soit de diminuer le nombre de fils requis tout en conservant la même bande passante. Cette dernière idée permet non seulement de réduire la surface de l'interconnexion mais aussi de réduire les effets du *crosstalk* et donc d'optimiser la consommation d'énergie et la fiabilité du lien. Afin d'améliorer la bande passante ou de diminuer le nombre de fils, le codage quaternaire est utilisé. Le principe est de coder quatre états sur un seul et même fil. Ceci se fait par l'ajout de deux niveaux de tension par rapport au codage binaire.

Pour l'optimisation de la consommation d'énergie, une approche consiste à implémenter des communications asynchrones. En binaire, ce type de communications est implémenté en utilisant le codage dual-rail [25], qui nécessite un codage sur deux bits, mais n'utilise que trois combinaisons. Une implémentation efficace peut être réalisée en utilisant le codage ternaire, ce qui réduit à un fil cet encodage tout en conservant les avantages de ce type de communications. Le dernier avantage de la logique MVL vient du fait que l'activité des signaux est jusqu'à 25% [66] plus faible qu'en logique binaire, cette propriété est très intéressante du point de vue de la consommation de puissance.

Ces avantages dépendent cependant de notre capacité à concevoir des codec efficaces. En effet, le principe est de conserver les parties calculatoires en binaire et de réaliser les transferts de données en MVL.

B.1-1 Présentation de la technologie SUS-LoC

Nous avons utilisé dans ce travail des transistors à seuil adapté. La faisabilité de ces transistors est assurée par une technologie SOI [1] de l'Université Catholique de Louvain [31]. La modification des seuils est assurée en agissant sur les différents paramètres de dopage. Deux circuits prototypes ont été réalisés avec cette technologie lors d'autres recherches dans l'équipe [44].

La conception des encodeurs (binaire vers MVL) utilise des transistors CMOS standards. Le décodage s'appuie sur une banque d'inverseurs avec des seuils de déclenchements différents. Le calcul des seuils de déclenchement V_{TH} des transistors est déterminé par les équations 3-1 et 3-2 respectivement pour les transistors PMOS et NMOS [64].

$$V_{TH}(PMOS) = Vi - (Vo - (OP \times LSV)) \tag{3-1}$$

$$V_{TH}(NMOS) = Vi - (Vo + (OP \times LSV)) \tag{3-2}$$

Vi est le niveau de la tension d'entrée à laquelle le transistor doit répondre, Vo est la tension de sortie. OP est le pourcentage de recouvrement, réglé à 70% dans nos travaux, alors que LSV est le pas entre deux niveau de tension.

Transistor	V_{TH} (V) quaternaire	V_{TH} (V) ternaire
Pm	-0.52	
Nm	0.52	
P+	-0.92	-0.78
N+	0.92	0.78
P-	-0.12	-0.18
N-	0.12	0.18

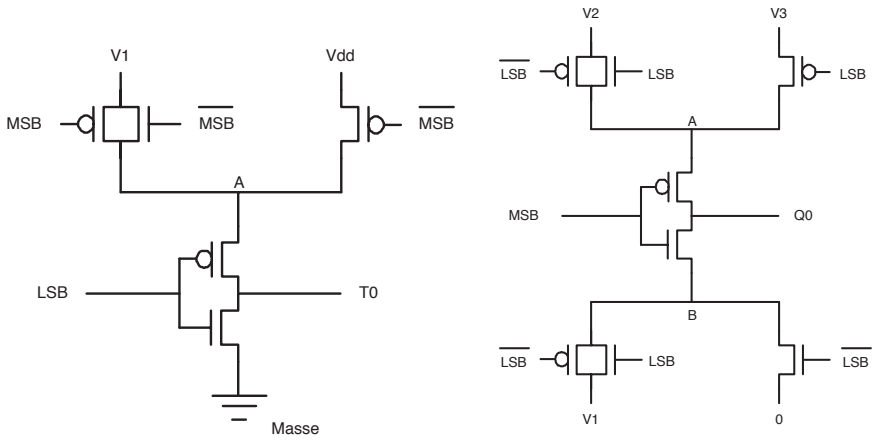
Tableau 3-2 – **Tensions de seuil modifiées des différents transistors ($V_{DD} = 1,2V$).** *Le codage quaternaire nécessite deux transistors de plus que le codage ternaire.*

1. Silicon-On-Insulator

La table 3-2 donne les tensions de seuil des différents transistors nécessaire pour une technologie 0.13 μm avec une tension d'alimentation de $1.2V$. On notera que la variabilité du process n'influe pas sur le fonctionnement des transistors dans une certaine mesure. Par contre, elle a un impact sur la marge de bruit en décalant les seuils des comparateurs.

B.1-2 Description des circuits

Encodeur m-valué L'encodeur permet de convertir le signal binaire en un signal m-valué.



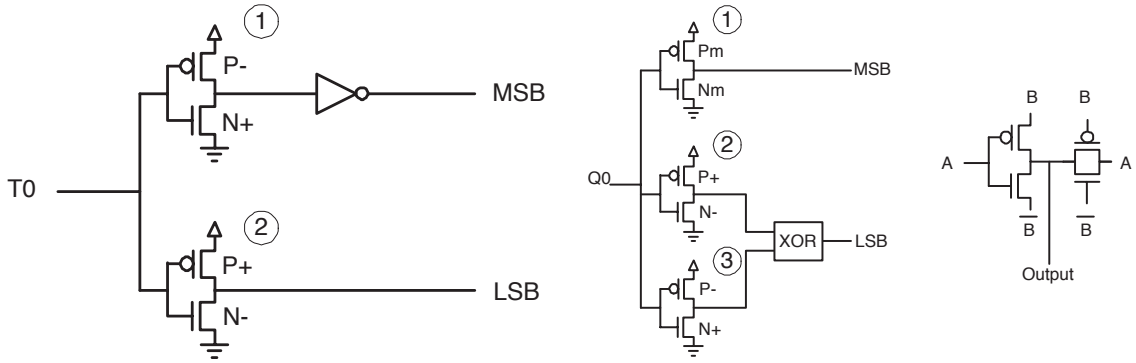
(a) Encodeur ternaire (b) Encodeur quaternaire

Figure 3-1 – **Schéma des encodeurs**. *L'encodeur ternaire nécessite 5 transistors et 8 transistors sont requis en quaternaire. Ces transistors ne sont pas modifiés. Il faut cependant rajouté un inverseur à chaque encodeur (obtention de $\overline{LSB}$ et de $\overline{MSB}$).*

L'encodeur ternaire (figure 3-1a) est optimisé pour réduire le nombre de transistors utilisés. L'élément central de décision donnant la valeur du signal ternaire est un inverseur commandé par le signal binaire LSB. L'encodeur quaternaire (figure 3-1b) est composé d'un inverseur central commandé par le MSB, les branches du haut et du bas sont composées d'interrupteurs commandés par le LSB. Ces deux encodeurs nécessitent un inverseur supplémentaire pour la conversion, le nombre total de transistors requis est respectivement de 7 pour l'encodeur ternaire, et 10 en quaternaire.

Décodeur m-valué Les décodeurs sont conçus pour délivrer deux signaux binaires à partir d'un signal m-valué. Leur conception nécessite une grande attention pour tirer partie des avantages de la MVL.

De par sa conception, le décodeur ternaire (figure 3-2a) ne nécessite que 6 transistors (4 transistors modifiés et un inverseur standard). De plus, il n'a pas de consommation de court-circuit car les inverseurs formés par les transistors à seuil modifié sont soit ouverts, soit fermés pour les trois tensions appliquées à leur entrée. A ma connaissance, c'est actuellement le plus petit décodeur.



(a) Décodeur ternaire (b) Décodeur quaternaire

Figure 3-2 – **Schéma des décodeurs**. *Le décodeur ternaire utilise un encodage avec retour à zéro. Les transistors ont des seuils de déclenchements modifiés (cf. table 3-2). Le XOR du décodeur quaternaire nécessite quatre transistors.*

Le décodeur quaternaire utilise quatre composants élémentaires, à savoir trois inverseurs réalisés avec des transistors à seuil modifié et une porte XOR en logique d'interrupteurs (figure 3-2b). Un inverseur supplémentaire est requis dans ce circuit. Ce décodeur utilise 12 transistors (6 modifiés et 6 standards) : sa surface est donc très petite.

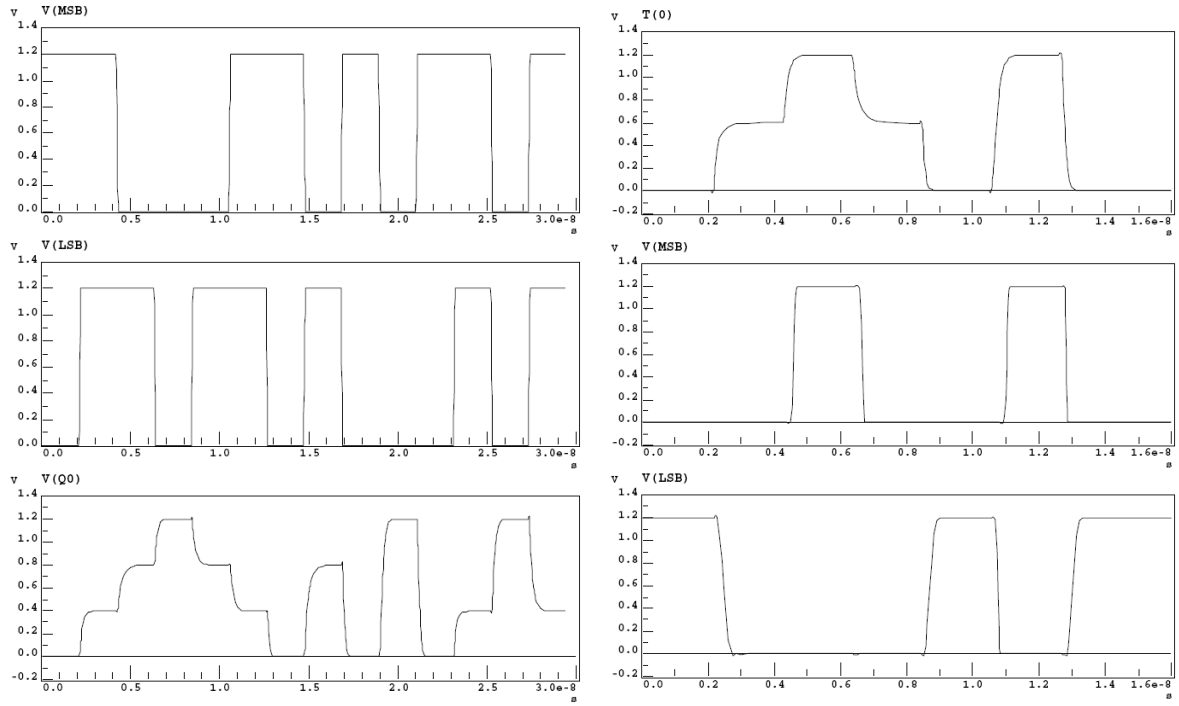
Les décodeurs proposés, de par leur simplicité, permettent de supporter de très hautes fréquences de fonctionnement.

B.1-3 Performances

Comme nous l'avons dit précédemment, une liaison ternaire est une alternative intéressante pour l'implémentation de communications asynchrones. Elle permet en effet de réduire la consommation du protocole dual-rail, ce qui est très intéressant pour la conception de circuits faible consommation. L'objectif principal du codage quaternaire est, quant à lui, de réduire le nombre de fils pour la transmission des données.

Dans nos expérimentations, tous les transistors ont des dimensions standards de $L = 2\lambda$ et $W = 12\lambda$ pour les PMOS et $W = 6\lambda$ pour les NMOS pour une technologie UMC 0.13 μm. Des optimisations sur ces dimensions peuvent être envisagées. Les encodeurs et les décodeurs sont reliés par un fil de métal-2 selon le modèle $\pi 3$ [76]. Les niveaux de tensions intermédiaires sont équiréparties entre la masse et V_{dd}. Tous les signaux d'entrées ont une fréquence de 500MHz. La consommation énergétique des différents circuits a été mesurée sous SPICE en envoyant une suite aléatoire de 1000 bits par entrée d'encodeur (MSB et LSB). La figure 3-3 montre l'exemple de la liaison quaternaire avec un fil de $1mm$ entre l'encodeur et le décodeur.

Nous avons mesuré les temps de propagation sur un lien complet incluant un fil de $1mm$. Pour le système quaternaire, le délai de propagation est de 1.18 ns. La contribution de l'encodeur est de 0.256 ns et 0.680 ns pour le décodeur. Comparé à d'autres implémentations [81][85], notre solution permet d'améliorer ce temps de propagation d'un facteur 1,8. Pour le système ternaire,



(a) Encodage (b) Décodage

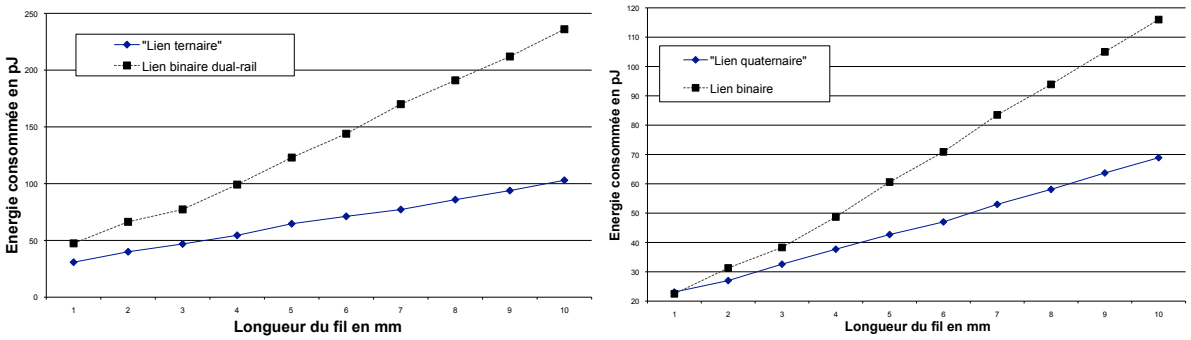
Figure 3-3 – **Liaison quaternaire**. *Signaux d'entrée et de sortie de l'encodeur et du décodeur quaternaire avec un fil de 1 mm.*

nous avons un temps de propagation de 724 *ps*, ce qui est extrêmement rapide dû au fait de la simplicité de la structure proposée. Afin d'évaluer l'impact de la technologie, nous avons aussi simulé un système où la taille du transistor PMOS de l'inverseur central a été doublé. Cette simple modification donne un temps de propagation de 539 *ps*, ce qui représente un gain de 25%. Il est à noter que ce gain augmente avec l'augmentation de la longueur du fil.

La consommation des systèmes ternaires a été comparée avec celle d'un système binaire asynchrone en *full-swing signaling* (figure 3-4a) composé de deux lignes (utilisant un codage double-rail) comprenant un inverseur standard en guise de *driver*, et un fil modélisé par le modèle $\pi 3$. La consommation des systèmes quaternaires a été comparée avec celle d'un système binaire en *full-swing signaling* (figure 3-4b) composé de deux lignes dans les mêmes conditions que pour le système ternaire. L'ensemble des circuits ont une charge de sortie correspondant à un inverseur standard.

Lorsque l'on compare les consommations d'énergie (figure 3-4), on relève des gains allant jusqu'à 56% pour le lien ternaire, et 43% pour la liaison quaternaire. La consommation énergétique d'un système m-valué est inférieure à un système binaire dont l'amplitude est égale à la tension d'alimentation (autrement dit, par rapport à un système en *full-swing signaling*). Cet avantage est tout à fait relatif et n'est plus vrai dans le cas d'un système avec une tension sur le fil inférieure à la tension d'alimentation (i.e. système en *low-swing signaling*). Cependant, décroître

la tension d'alimentation dans les deux cas équivaut à augmenter la probabilité d'erreurs.



(a) Lien ternaire (b) Lien quaternaire

Figure 3-4 – **Consommation des liaisons m-valué**. *La consommation d'énergie est donnée en fonction de la longueur du fil pour une technologie 0.13 µm. L'énergie du lien ternaire est comparée à une liaison binaire asynchrone en dual-rail. La liaison quaternaire est comparée à la liaison binaire équivalente (deux fils).*

Cependant, la diminution de la surface d'interconnexion est un avantage intéressant. Le nombre de fils nécessaires à la transmission du signal est divisé par deux en MVL. Cet avantage permet alors d'écarter les fils et de diminuer ainsi l'effet du *crosstalk*, ce qui rend la liaison plus fiable et plus rapide. L'utilisation de la MVL permet en outre de diminuer l'activités des signaux. En conclusion les technologies MVL offrent de réelles opportunités pour la conception d'interconnexions efficaces, mais nécessitent un processus de fabrication complexe. Nous avons alors travaillé sur des systèmes actifs de codages.

B.2 Codage Spatio-temporel

Notre premier codage conjoint vise à augmenter la fiabilité de la liaison tout en optimisant la consommation des interconnexions. Les effets du *crosstalk* sont réduits par le décalage et l'entrelacement des bits de deux données indépendantes (figure 3-5). Ce décalage d'une demi période d'horloge de transmission, élimine donc les transitions de pire cas de la table 3-1, les seules transitions restantes sont $(-,\uparrow,-)$ et $(-,\downarrow,-)$. De ce fait, le facteur de délai g est grandement réduit. Cette technique à les mêmes résultats que la duplication ou le *shielding*, sauf que les fils rajoutés servent aussi à la transmission, ce qui n'est pas le cas des deux techniques précitées.

Afin d'améliorer la tolérance aux bruits nous utilisons deux techniques. La première consiste à utiliser une horloge de réception deux fois plus rapide que l'horloge de transmission. En effet, d'un point de vue temporel cette dernière est forcément plus lente vu que nous utilisons les deux phases de ce signal. De fait, la fréquence de la réception peut être doublée. A la réception on obtient donc deux échantillons d'une même donnée. Nous avons alors une détection d'erreur

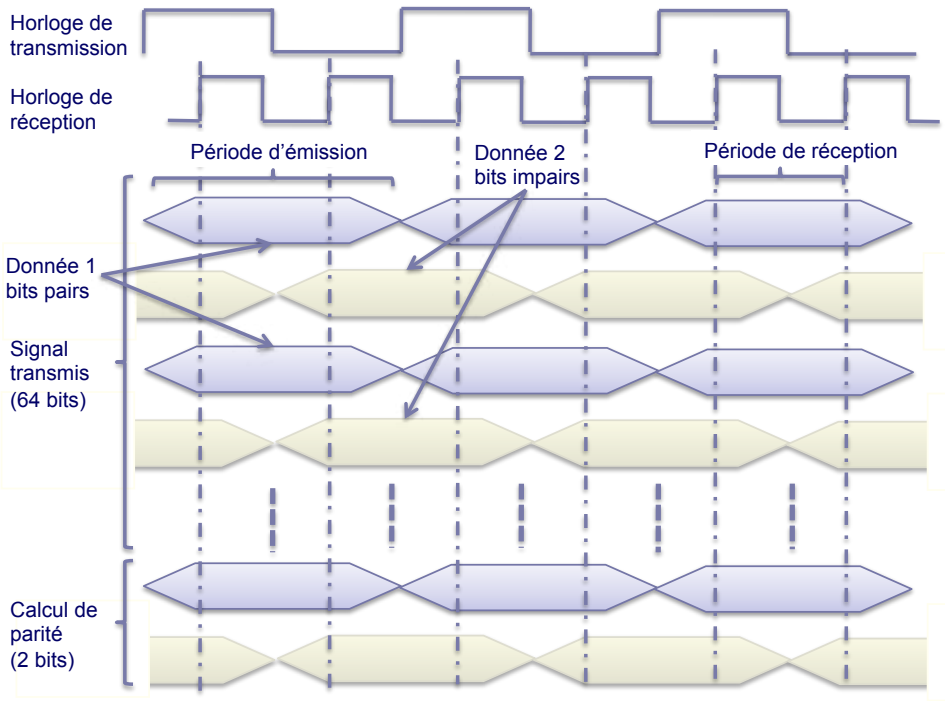


Figure 3-5 – **Chronogramme du bus de transmission dans notre système de codage conjoint pour des mots de 32 bits**. *Les signaux impairs sont décalés par rapport aux signaux pairs (décalage entre les bits du mot 1 et les bits du mot 2) éliminant les transitions pire-cas. L'horloge de réception est deux fois plus rapide que l'horloge de transmission permettant un suréchantillonage des données. Des bits de parité sont envoyés sur le bus de la même façon que les données afin de ne pas ralentir la transmission.*

temporelle par simple comparaison des deux échantillons. La seconde technique utilise le calcul de bits de parités pour chacun des mots transmis (deux derniers bits de la figure 3-5). Ce bit est transmis en même temps que la donnée correspondante, et est une technique très simple de détection d'erreur. On notera qu'il est tout à fait possible d'implémenter des codages plus efficaces au prix d'une complexité plus grande.

Ce "codage" peut être mis en œuvre à très faible coût en terme de surface. L'encodeur est très simple car la technique utilisée ne nécessite pas de gros traitements. Le codage atténuant le *crosstalk* choisi pour ce système ne nécessite aucun composant supplémentaire à l'émission comme à la réception. Le décalage d'une demi période est réalisé en rendant sensible l'émetteur/récepteur pair à un front montant, et l'émetteur/récepteur impair à un front descendant de l'horloge de transmission/réception alternativement. Le codage contre le bruit à l'émission consiste tout simplement en un encodeur de parité réalisé à l'aide d'un arbre de portes XOR. On peut noter que l'émetteur fonctionne à la vitesse d'horloge la plus lente. La consommation est donc réduite pour le chargement des capacités formées par les fils du bus. Le décodeur est quant à lui plus complexe. La réception est réalisée au double de la vitesse d'émission sur le bus, afin de réaliser un double échantillonnage. Le décodeur possède un registre à décalage de

deux places par fil, ces deux places étant reliées à une porte XOR pour la détection temporelle. En cas de détection d'erreurs soit temporelle, soit par calcul de la parité une retransmission est demandée.

Les mesures effectuées, ont montré que l'élimination des pires cas de transmission permettent une augmentation du débit de la liaison par 2,3 en doublant le nombre de fils. Nous avons aussi évalué la probabilité d'erreurs résiduelles en fonction de la tension d'alimentation du circuit (figure 3-6).

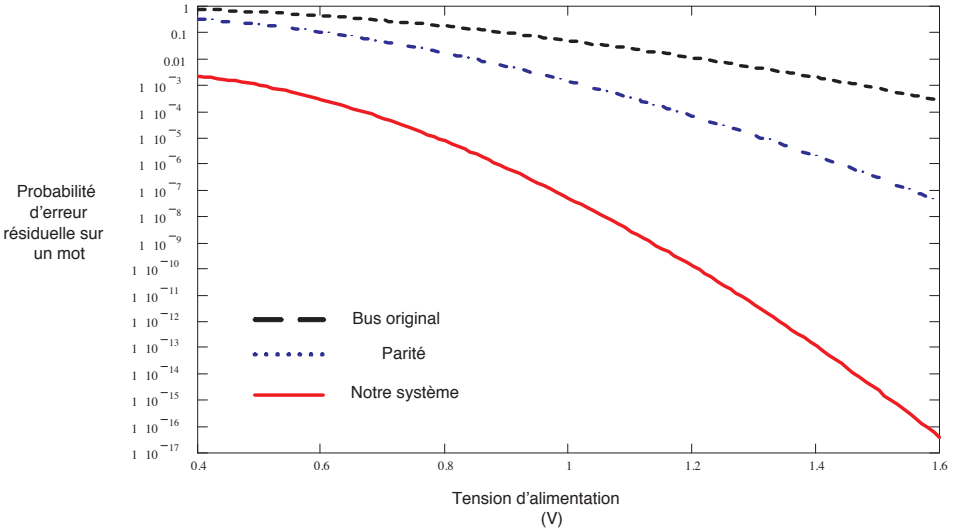


Figure 3-6 – **Calcul des erreurs résiduelles**. *Superposition des courbes de probabilité d'erreurs résiduelles sur un mot avec un bus non protégé, un système de parité et notre système pour une transmission de mots de 8 bits.*

Nous pouvons dire que notre système est beaucoup plus performant en détection d'erreurs que le système de parité. En effet, le débit de notre système est supérieur au débit d'un bus avec un bit de parité et la variance du bruit sera moindre. Dans ce contexte, pour une probabilité de détection d'erreurs donnée, nous pouvons réduire la tension d'alimentation du circuit afin d'optimiser la consommation d'énergie de la transmission. Par exemple, comme on peut le voir sur la figure 3-6 pour une même probabilité d'erreur résiduelle de 10^{-4} obtenue avec un calcul de parité à une tension de $1,2V$, nous pouvons réduire la tension d'alimentation à $\approx 0.7V$. Notre codage permet donc une augmentation du débit, une amélioration de la fiabilité et la réduction de la consommation d'énergie.

B.3 Codage pour canaux asymétriques

Un canal asymétrique de communications est un modèle dans lequel la probabilité qu'un niveau logique 1 se transforme en niveau logique 0 est beaucoup plus grande que l'inverse. Ce type de canaux se retrouve dans les communications optiques, mais aussi dans les mémoires ROM

flash, SRAM et dans des dispositifs conçus pour la tolérance aux fautes ou l'optimisation de la consommation. Ce type de canaux apparait donc de plus en plus avec les besoins d'optimisations actuels en termes de consommation, de tolérance aux fautes, ou même encore d'accès à des mémoires caches.

Récemment, Huang [39] a proposé de combiner un code de Berger [13] avec une technique d'inversion de bus (BI – Bus Invert) [83] afin de détecter les erreurs dans un canal asymétrique mais aussi de réduire la consommation de puissance de la liaison. Nous avons alors proposé une amélioration de ces travaux, en utilisant de manière optimale le codage de Berger.

Les codes de Berger sont optimaux quant à la détection de n'importe quelles erreurs asymétriques. Le principe est de rajouter des bits de contrôle représentant le nombre de 0 ou le nombre de 1 dans la donnée de départ. Cette information est alors utilisée à la réception pour détecter d'éventuels changements de valeurs. L'inversion de bus, quant à lui, consiste à émettre les données avec le moins de un possible dans la donnée émise. L'objectif principal est de réduire l'activité des signaux et donc de réduire la consommation d'énergie, mais aussi la probabilité d'erreurs dans le canal asymétrique. Ceci induit l'ajout d'un bit (I) permettant de signaler si la donnée émise est inversée (au sens du complément à 1). Le code VHDL de l'encodeur est présenté sur le listing 3-1. Le décodeur a la même structure sauf qu'il compare le nombre de 1 (ou de 0) trouvé avec celui reçu pour détecter les erreurs de transmissions.

Les codec ont été synthétisés avec les outils *Synopsys Design compiler* pour une technologie ST 90 nm avec un $V_{dd} = 1V$. Les optimisations architecturales que nous avons apportées vis-à-vis de la solution présentée dans [39], permettent un gain en surface de $\approx 15\%$ pour un bus 8 bits et jusqu'à $\approx 43\%$ pour un bus 32 bits. Nos codec sont donc très efficaces pour le même résultat en terme de fiabilité de la liaison. Nous avons alors évalué le gain en consommation des différentes approches. Pour ces estimations, nous avons généré avec MATLAB un ensemble aléatoire de 100000 échantillons. Nous avons synthétisé l'ensemble des circuits pour la même technologie avec une contrainte fréquentielle de 100 MHz (contrainte facilement supportée pour tous les circuits). L'activité des codec a été obtenue post-synthèse avec Modelsim SE Plus 6.4b pour la bibliothèque nominale. La consommation a alors été évaluée grâce à l'outil Prime-Time B-2008.O6 de Synopsys.

La table 3-3 montre les résultats de consommation pour les liaisons complètes (incluant les fils). Nous avons choisi un fil de 2 mm de métal-4. L'estimation de la consommation des fils a été obtenue avec l'outil *Interconnect Explorer* [22] développé dans l'équipe. Le gain entre notre approche et celle proposée dans [39] est calculé par $Red = (P_{BGI_{Huang}} - P_{BGI_{New}})/P_{BGI_{Huang}}$.

La consommation des fils seuls est équivalente dans les deux approches : la différence de consommation se fait sur la conception des circuits. Les simplifications que nous avons proposées permettent de réduire pratiquement la consommation par deux pour des bus de 30 bits et au delà. On notera que le gain pour une taille de donnée de 7 bits n'est pas bon, car cela correspond au pire cas pour notre approche.

```
1  entity berger_enc is
2      generic (
3         m : integer := 4;                -- taille du code de Berger
4         n : integer := 12);              -- taille de la donnée
5      port ( data : in Std_Logic_Vector(n-1 downto 0);
6         clk : Std_Logic;
7         edata : out Std_Logic_Vector(m+n downto 0) ); --taille sortie=(n+m+1)
8  end berger_enc;
9
10 architecture behavioral of berger_enc is
11     signal d : std_logic_vector(n+m downto 0) := (others => '0');
12     signal NbUn : std_logic_vector(m-1 downto 0) := (others => '0');
13     signal NbZero : std_logic_vector(m-1 downto 0) := (others => '0');
14 begin
15    --Process de comptage du nombre de un non montré pour des raisons de place
16    . . .
17     process (data,NbUn, NbZero)
18        variable I : std_logic;
19        variable codeBerger : std_logic_vector(m-1 downto 0);
20        begin
21           if NbUn > NbZero then
22              I:='1';
23              codeBerger := NbUn;
24           else
25              I:='0';
26              codeBerger := NbZero+1; -- +1 pour inclure I dans la détection
27           end if;
28           for k in n-1 downto 0 loop
29              d(k) <= data(k) xor I; -- BI de donnée en fonction de I
30           end loop;
31           d(n) <= I; -- rajout de I dans la donnée transmise
32           d(n+m downto n+1)<= codeBerger; -- rajout des bits du code de Berger
33     end process;
34     process(clk)
35        begin
36           if clk'event and clk='1' then
37              edata <= d;
38           end if;
39     end process;
40 end behavioral;
```

Listing 3-1 – ***Description VHDL de l'encodeur de Berger proposé.*** *Après avoir déterminé le nombre de 1 et de 0 dans la donnée (non représenté ici), la donnée transmise est construite avec la donnée originale, inversée ou pas, le bit du BI et le code de Berger.*

En conclusion, pour une même protection de la liaison, notre approche permet de diminuer de $\approx 40\%$ la surface des codecs en diminuant par deux la puissance consommée pour un bus de 32 bits. Le coût de cette protection est cependant élevé en surface et en puissance (5x plus de consommation vis-à-vis d'une liaison simple).

Taille code (m)	Taille données (n)	simple (μW)	BGI$_H$ (μW) [39]	BGI$_N$ (μW)	BGI$_H$ vs simple	BGI$_N$ vs simple	Red. (%)
3	6	72.88	153.20	144.08	2.10	1.97	6.33
4	7	90.57	196.78	191.51	2.17	2.11	2.75
	10	129.87	307.27	275.27	2.36	2.12	11.62
	14	175.76	487.22	415.36	2.77	2.36	17.30
5	16	203.93	1093.33	758.60	5.36	3.72	44.12
	30	380.16	3925.19	2059.03	10.32	5.41	90.63
6	32	403.7	4419.37	2296.49	10.96	5.69	92.44

Tableau 3-3 – **Consommation de puissance du codage pour différentes tailles de données**. *Ces résultats incluent les fils. L'approche simple correspond à une liaison sans codage.*

C Couches transport et réseau

Contexte
Les travaux sur les réseaux sur puce sont menés dans les thèses de Ludovic Devaux et de Istas Pratomo. Une partie de ces travaux est utilisée dans le cadre du projet FosFor. L'objectif est de définir et d'implémenter dans une ARD une architecture d'interconnexion générique afin de supporter la diversité des applications et la gestion dynamique des tâches.

Au travers de l'architecture physique d'interconnexion et de son contrôle, le service de communication d'une ARD doit permettre une communication flexible entre les différents éléments communicants (EC) d'une application. Le terme EC désigne alors les éléments matériels qui produisent/reçoivent des données. Ainsi, un EC peut être l'implémentation matérielle d'une tâche (statique ou dynamique), d'un élément partagé (mémoire, entrée/sortie, etc.), ou encore d'un processeur matériel qui exécute des tâches logicielles.

Pour correspondre à un large spectre d'applications implémentées sur une ARD, une architecture d'interconnexion doit pouvoir répondre à un certain nombre de contraintes. Tout d'abord, les applications actuelles sont très complexes et leurs décompositions en tâches montrent un degré croissant de parallélisme. Au niveau de l'interconnexion, il est donc important de pouvoir réaliser plusieurs communications en parallèle. Ensuite, le placement et l'ordonnancement dynamique des EC dans un FPGA requièrent un niveau élevé de flexibilité. En effet, ni la localisation des EC, ni le trafic de données (uniforme, rafale, etc.) ne peuvent être prédit au moment de la compilation. Ce besoin de flexibilité doit donc être supporté par la topologie du réseau, le protocole de routage, et les performances disponibles (bande passante et latence). Comme nous l'avons dit, une application est typiquement décomposée en un ensemble de tâches statiques et dynamiques, matérielles et logicielles, et il n'y a aucune raison pour qu'elles soient implémentées sous forme d'EC de tailles homogènes. Ce point diffère des approches comme [23] où les tâches sont supposées être de tailles homogènes. Ainsi, en considérant les contraintes de placement du réseau et des EC, une topologie réseau compatible avec des tâches hétérogènes

sera préférée à des topologies dynamiques.

A l'heure actuelle, seuls les FPGA Virtex de Xilinx supportent la reconfiguration dynamique partielle [91] et proposent un nombre important de ressources programmables. Ces architectures offrent une distribution en colonne de leurs ressources, comme présenté dans la figure 3-7. Dans les gammes Virtex (excepté le Virtex 2pro), la colonne centrale est très spécifique car composée de banques d'entrées/sorties (IO), de gestionnaires d'horloges, et de ports de reconfiguration (ICAP). Ainsi, du fait de leur structure en colonne, les FPGA Xilinx sont très hétérogènes du point de vue de l'implémentation en 2 dimensions (ligne/colonne) des tâches. Par conséquent, l'architecture d'interconnexion doit supporter l'hétérogénéité de ces circuits.

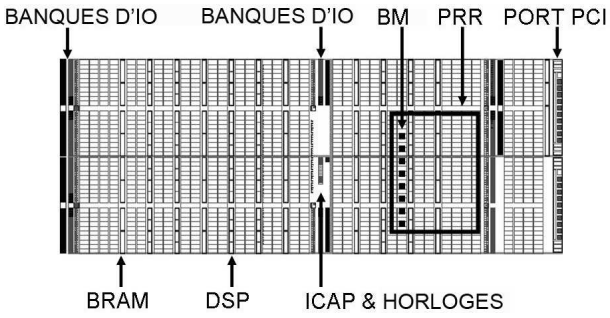


Figure 3-7 – **Vue simplifiée d'un FPGA Xilinx Virtex5 XC5VSX50T**. *Les architectures Xilinx sont basées sur une construction en colonne. Un ensemble de ressources hétérogènes constitue la matrice de FPGA.*

La reconfiguration dynamique d'un FPGA Xilinx a lieu à l'intérieur de régions reconfigurables appelées PRR (*Partially Reconfigurable Regions*). Les PRR forment les parties dynamiques d'une architecture dans lesquelles les EC dynamiques sont alloués. L'extérieur des PRR forme la partie statique du système et c'est là que sont implémentés les EC statiques (tâches statiques, mémoires, processeurs, etc.). Les communications entre les régions dynamiques et la partie statique sont réalisées au travers d'interfaces appelées *Bus Macro* (BM) dans les architectures Xilinx [88]. PRR et BM sont définis statiquement au moment de la compilation. Ainsi, tout EC est connecté à l'architecture d'interconnexion via une interface statique, même si l'EC lui-même est dynamique. Par conséquent, l'architecture d'interconnexion peut être statique si elle supporte les besoins en matière de flexibilité.

La reconfiguration dynamique partielle implique donc du parallélisme de communications, de la flexibilité et la compatibilité avec une connexion d'EC hétérogènes dans un FPGA hétérogène.

Des approches à base de bus ont été proposées [2] [8] [45]. Le problème majeur de ce type d'approches vient du passage à l'échelle, et du fort surcoût matériel nécessaire pour supporter des communications parallèles. Il existe actuellement de nombreux travaux sur la définition de NoC [77], mais très peu supportent et utilisent la reconfiguration dynamique. La topologie mesh est la plus utilisée dans ce cadre [15][1][21] [41]. Le point faible de cette topologie réside dans sa régularité intrinsèque. De ce fait, des ressources logiques sont gaspillées selon la granularité des EC. En considérant la structure hétérogène des FPGA actuels, il semble difficile d'implémenter

un mesh vraiment régulier. Enfin, un autre problème de ces structures réside dans l'interconnexion d'éléments partagés comme des mémoires ou des entrées/sorties. En effet, les besoins de communication entre les tâches et ces éléments partagés induisent la création de points chauds (*hot-spots*) qui accroissent la probabilité d'apparition de contentions (*livelock*) et de congestions (*deadlock*). Une approche intéressante est alors d'adapter la topologie selon l'application, soit à priori [23], soit dynamiquement [72]. Dans la première approche, la création du réseau nécessite de connaître à priori toutes les futures tâches de l'application et leur placement. Dans la deuxième approche, l'utilisation de PRR pour implémenter des liens de communication implique un gaspillage de ressources logiques. De plus, dans ces réseaux, le protocole de routage doit prendre en compte la topologie du réseau qui elle même dépend de l'application, ce qui peut engendrer des temps de conception prohibitifs.

La compatibilité entre les architectures que nous avons étudiées et les contraintes liées à la reconfiguration dynamique partielle est résumée dans le tableau 3-4.

Contraintes	bus	mesh	ad'hoc
Ordonnancement dynamique	NON	OK	OK
Placement 2D	NON	OK	OK
EC hétérogènes	OK	NON	OK
FPGA hétérogènes	OK	NON	NON
Ressources utilisées	faible	moyen	haut
Flexibilité	faible	haute	très haute
Parallélisme des communications	faible	haut	haut
Généricité	moyenne	haute	faible

Tableau 3-4 – **Compatibilité entre les architectures d'interconnexions actuelles et la reconfiguration dynamique partielle**.

Comme nous pouvons le voir, aucune étude actuelle ne répond à l'ensemble de la problématique de l'interconnexion au sein d'une ARD. Lors de notre analyse, il est apparu que la topologie en arbre (notamment le fat-tree) a un grand intérêt dans ce contexte. Le fat-tree offre de nombreux avantages par rapport aux autres topologies dont une bande passante large et une latence faible [67]. Chaque EC étant connecté à un routeur situé à la base du fat-tree, ils ne sont pas distribués dans la structure du réseau. Ce point est important car les EC peuvent donc être de tailles hétérogènes. De plus, la structure d'un fat-tree n'a pas besoin d'être implémentée de manière régulière. Ainsi, un fat-tree est compatible avec l'hétérogénéité des EC et des FPGA. Cependant, sa consommation de ressources logiques reste son principal défaut. C'est pourquoi, cette étude des architectures d'interconnexion conduit au réseau DRAFT (*Dynamic Reconfiguration Adapted fat-tree*).

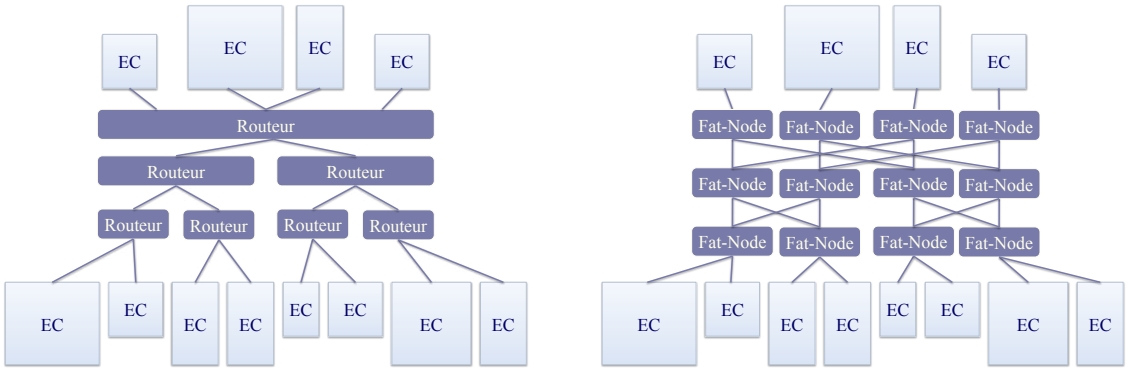
C.1 Interconnexion flexible : DRAFT

De la comparaison des NoC actuels, le fat-tree apparaît comme étant le réseau le plus apte à supporter des applications utilisant la reconfiguration dynamique partielle des FPGA. DRAFT

est un réseau basé sur la topologie fat-tree dont la principale caractéristique réside dans la réduction du nombre de ressources logiques utilisées pour le routage.

C.1-1 Topologie DRAFT

Le concept proposé dans DRAFT est de connecter directement des EC au sommet d'un fat-tree (figure 3-8a). Ce concept réduit de manière significative le nombre de ressources logiques utilisées pour le routage par rapport au nombre d'EC connectés. En effet, pour un même nombre d'EC connectés, le nombre de routeurs est divisé par deux par rapport à une topologie fat-tree.



(a) Vue routeur (b) Vue Fat-Node

Figure 3-8 – **La topologie DRAFT**. *Les éléments communicants "EC" sont connectés à la base et au sommet d'un réseau de type fat-tree. La vue Fat-Node est construite autour d'un ensemble de routeurs génériques.*

Il y a différentes façon de voir un fat-tree et donc DRAFT. Ainsi, les deux arbres présentés dans les figures 3-8a et 3-8b ont les mêmes propriétés vis à vis des EC. En effet, un routeur de la figure 3-8a peut être décomposé en un ensemble de routeurs unitaires que l'on appelle Fat-Node (figure 3-8b). Cela permet de construire un réseau en utilisant un routeur générique, et donc d'en faciliter la génération automatique. Cependant, la structure Fat-Node n'est pas complètement compatible avec la connexion d'EC au sommet de DRAFT car elle réduit les possibilités de routage. Ainsi, la structure Fat-Node est moins flexible que la structure routeur mais est plus générique. Nous avons cependant considéré cette structure dans un premier temps afin de profiter de ses routeurs génériques. Une évolution de DRAFT vers la structure routeur permettra de supporter des applications nécessitant plusieurs chemins de données entre des EC connectés au sommet et à la base de l'arbre.

Concernant les performances réseau, la distance entre les EC connectés au sommet et à la base de l'arbre est constante quelles que soient leurs localisations respectives. Ainsi, la latence des communications entre ces éléments est constante et minimale. Cependant, pour que cette topologie soit efficace, il est nécessaire que les EC connectés au sommet ne communiquent qu'avec ceux connectés à la base de l'arbre. Cette hypothèse empêche la création de *hot-spot* dans le routeur au sommet de l'arbre. Des communications entre les EC connectés en haut de

l'arbre requerraient des niveaux hiérarchiques supplémentaires (conduisant à la topologie fat-tree) afin de ne pas augmenter la charge du routeur du sommet. Cette hypothèse est importante mais tant qu'elle est respectée, il n'y a en fait aucune limitation concernant la nature des EC connectés en haut de l'arbre. Un développeur est donc libre de connecter des éléments partagés au sommet de DRAFT, mais aussi des tâches matérielles (statiques ou dynamiques), ou même des processeurs. Ainsi, moyennant cette hypothèse, DRAFT est complètement flexible. Les EC connectés à la base de l'arbre peuvent, quant à eux, communiquer indifféremment avec tous les autres EC.

C.1-2 Stratégie de routage

Le réseau DRAFT utilise une architecture de routeur classique basée sur quatre ports de communication bidirectionnels, chacun incluant une FIFO asynchrone. Les tailles des FIFO sont définies par le développeur selon la largeur et le nombre de flits[2], et en fonction du flot de données et du protocole de gestion de contention (basé sur les crédit, sur les priorités, etc.).

Le routage des données est assuré par un algorithme de *Turn-Back* [43]. La prochaine destination d'un message est calculée par chaque routeur qui le reçoit. La décision est prise en utilisant un jeu de plusieurs masques sur les adresses de source et de destination du message, qui sont inclues dans son entête. Chaque routeur est identifié par une adresse interne qui indique son niveau hiérarchique et sa position dans ce niveau. De la même manière, tous les EC connectés à DRAFT sont identifiés par des adresses internes qui sont utilisées pour spécifier la source et la destination d'un message. Grâce à ces adresses, chaque routeur utilise un masque dépendant de son niveau hiérarchique afin de déterminer le sens de routage d'un flit. Ainsi, chaque donnée est routée vers le haut jusqu'à ce qu'elle soit capable de redescendre dans la moitié (ou sous moitié) désirée de l'arbre. Ce routage descendant est directement appliqué aux EC connectés au sommet de DRAFT. Les adresses de destination sont suffisantes pour déterminer vers quelle partie de l'arbre la donnée doit être routée. Cet algorithme garantit une distance minimale aux données et qu'il ne peut y avoir aucune congestion. Afin que l'architecture de DRAFT reste statique dans le cadre de la reconfiguration dynamique partielle, l'adressage des routeurs et des EC est générique. Cependant, comme beaucoup d'EC sont dynamiques, cela oblige le développeur à s'assurer qu'une table de routage est tenue à jour par le placeur/ordonnanceur de tâches. Cette table est essentielle aux interfaces réseaux pour faire la correspondance entre les adresses logique de l'application et les éléments physiques (tâches implémentées, mémoires, ...).

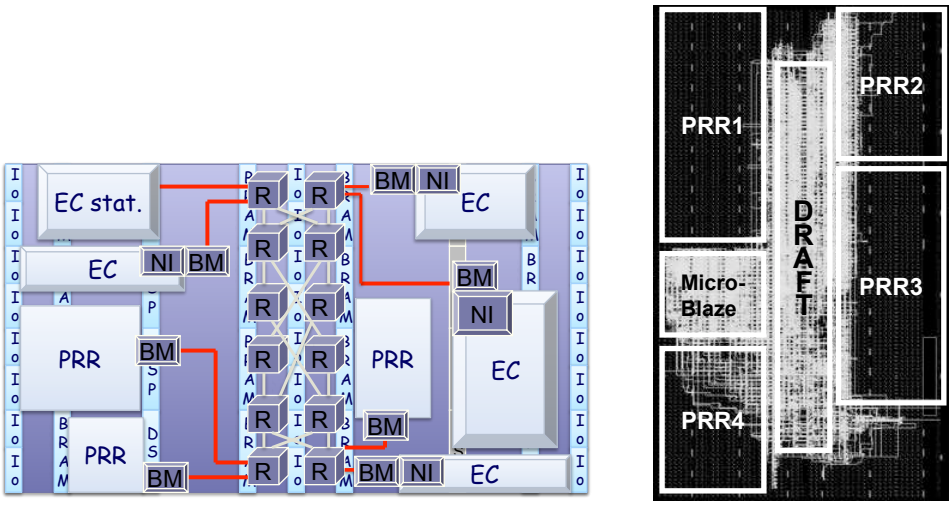
C.1-3 Adaptation à la reconfiguration dynamique

DRAFT est vu comme un cœur de réseau indépendant qui connecte les différents éléments de l'application, apportant la flexibilité requise par la reconfiguration dynamique partielle.

2. Les données sont fractionnées sur le réseau en petits paquets de bits appelés flits

Dans l'architecture globale, les EC sont tous implémentés dans un PRR. Il peut cependant y avoir des EC ne nécessitant pas de reconfiguration dynamique (interface avec la mémoire externe par exemple). Ces tâches statiques doivent alors être connectées à DRAFT. Pour cela, le développeur a le choix de les connecter directement à un routeur au même titre qu'une tâche dynamique, ou bien de les interconnecter via un bus ou un sous réseau lui-même connecté à un port d'un routeur de DRAFT.

Dans le cas des applications utilisant la reconfiguration dynamique, DRAFT ne connecte pas les EC directement au travers de leurs interfaces réseau mais via les *Bus Macros* (BM). Ainsi, les BM sont les interfaces entre DRAFT et les EC, qui de fait incluent leurs propres interfaces réseau (NI– Network Interface) comme présenté sur la figure 3-9a.



(a) Vue logique

(b) Vue physique pour 4 PRR et 4 IO partagées

Figure 3-9 – **Implémentation de DRAFT**. *Afin de tirer au mieux partie de l'architecture des FPGA Xilinx, le réseau est implémenté comme une colonne centrale interconnectant des EC qui sont alloués dans les deux moitiés du circuit. Les EC communiquent aux travers de NI adaptées, la connexion au réseau est assuré par des Bus Mcro (BM). Un EC statique est directement relié au routeur de DRAFT.*

Ce concept est important car les NI sont alors dynamiques et peuvent donc être conçues de manière optimale pour chaque EC. Cela permet notamment de réduire le coût matériel des NI lorsque les EC n'ont pas tous les mêmes interfaces. Ainsi, lors de la reconfiguration dynamique d'un EC, l'interface d'entrée de DRAFT reste la même, alors que l'interface du nouvel EC ainsi que sa NI sont adaptés. Cela permet d'augmenter la généricité de notre réseau, ainsi que sa flexibilité par rapport au placement des EC.

Le placement de DRAFT est important dans la conception d'un système utilisant la reconfiguration dynamique partielle car il impacte l'utilisation des ressources reconfigurables ainsi que les performances réseau. Ainsi, l'idée présentée sur la figure 3-9a est de l'implémenter comme une colonne centrale au FPGA. Ce concept est particulièrement adapté aux technologies actuelles

de FPGA Virtex. Les EC sont implémentés dans les deux moitiés du circuit avec les éléments statiques de l'application (processeur, etc.). DRAFT n'étant pas distribué dans le FPGA, le développeur n'est pas contraint par le réseau dans la définition des tailles et le positionnement des PRR. Cela constitue un avantage pour l'implémentation d'EC dynamiques hétérogènes. Ainsi, l'implémentation de DRAFT est pleinement compatible avec la technologie actuelle et les contraintes liées à la reconfiguration dynamique partielle (figure 3-9b).

Dans les FPGA Xilinx, les entrées/sorties partagées doivent, de préférence, être placées dans les banques centrales d'IO. De même, il est recommandé de placer les mémoires partagées dans les colonnes de BRAM les plus proches de la colonne centrale. Ces éléments sont connectés au sommet de DRAFT et les communications entre les tâches et ces éléments rencontreront une latence minimale.

C.2 Le générateur DRAGOON

DRAGOON (*Dynamically Reconfigurable Architecture compliant Generator and simulatOr Of Network*) est un environnement de conception spécialement conçu pour générer et simuler le réseau DRAFT, mais il supporte également la topologie fat-tree. Le flot de conception mis en œuvre par DRAGOON est illustré dans la figure 3-10.

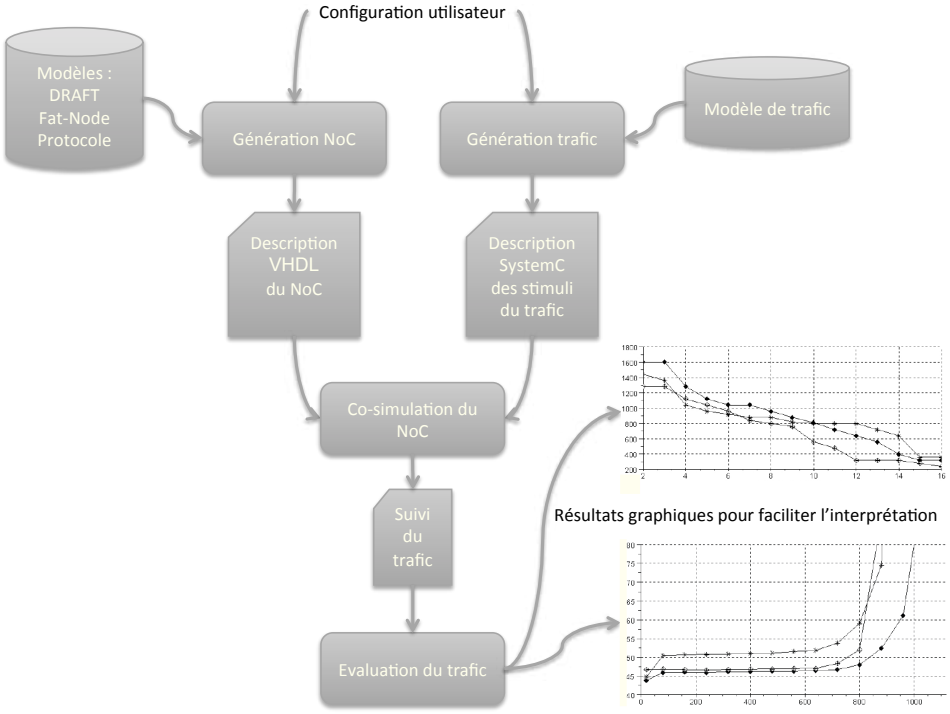


Figure 3-10 – **Flot de conception mis en œuvre par DRAGOON**. *Ce flot génère l'ensemble des fichiers nécessaires pour la simulation ou la synthèse du NoC. Différents générateurs de trafic sont disponibles pour les évaluations de performances.*

L'outil de génération de NoC produit la description VHDL de DRAFT ainsi que des *test*

benches écrits en SystemC, selon la configuration réseau choisie par le développeur. Ce dernier peut choisir la dimension du réseau en termes d'EC connectés, la taille des flits, ou encore la profondeur des buffers (FIFO). La taille des flits détermine la longueur des données qui sont échangées sur le réseau, et peut être fixée à 16, 32, ou 64 bits. La profondeur des buffers indique combien de flits peuvent être mémorisés dans chacune des quatre FIFO d'un routeur. Ainsi, un routeur peut stocker 4, 8, 16, ou 32 flits dans chacun de ses quatre ports. Le nombre de canaux virtuels peut aussi être spécifié par le développeur. En utilisant cette stratégie, un routeur est capable de supporter plusieurs communications en parallèle moyennant des ressources logiques supplémentaires. Le type de contrôle de flux (basé sur les crédits ou sur un acquittement des données) ainsi que la politique d'ordonnancement (circulaire ou par priorités) sont aussi paramétrables.

L'outil de génération de trafics produit différents trafics de données (*uniforme, normal, Pareto on/off*) en vue de l'évaluation des performances d'un réseau particulier. Chaque trafic simule un type d'application supporté par DRAFT. Un trafic *uniforme* simule des applications qui utilisent un flot constant de données comme le traitement vidéo. Un trafic suivant une loi de distribution *normale* est généré par les applications dont les actions dépendent des données, comme la reconnaissance de formes ou le suivi de cibles. Enfin, le trafic *Pareto on/off* désigne une communication entre une tâche et une mémoire partagée où les données sont transmises en utilisant un mode rafale (période ininterrompue de transmission de données suivie d'une période de silence). De plus, l'outil de génération de trafic permet au développeur de simuler de nombreuses configurations concernant la connexion des EC. En effet, les fréquences des EC, les destinataires des données émises (aléatoires ou spécifiques), le nombre de paquets à émettre, ainsi que le nombre de flits dans un paquet sont paramétrables. Le développeur peut aussi spécifier le débit de transmission de chaque EC. En utilisant tous ces paramètres, le générateur de trafics construit un fichier contenant toutes les données qui seront transmises au travers du réseau.

L'outil de simulation invoque le simulateur VHDL ModelSim. Cet outil a été choisi car il supporte la cosimulation VHDL et SystemC. Ainsi, l'outil de simulation utilise les descriptions VHDL et SystemC du NoC, ainsi que le fichier de trafic généré. Ce trafic est injecté dans DRAFT pendant la phase de simulation, au terme de laquelle les fichiers de sortie sont générés.

Enfin, l'outil d'évaluation apporte une interprétation des résultats grâce aux fichiers de sortie générés précédemment. Les résultats sont analysés et présentés sous forme de graphiques et de rapports. Les performances réseau comme la latence et le taux d'utilisation (*throughput*) sont les principaux résultats présentés par cet outil.

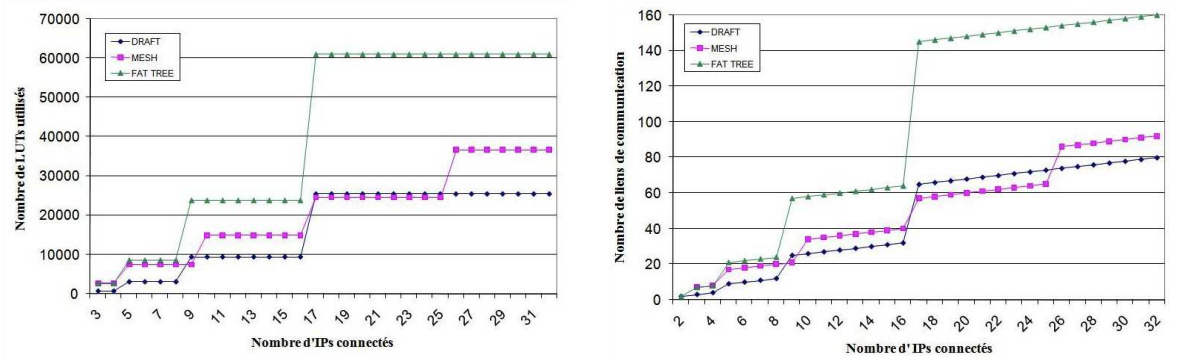
C.3 Résultats d'implémentation

Nous avons comparé DRAFT à un réseau de type mesh (Hermes [58]) et un fat-tree. Cette comparaison prend en compte l'utilisation des ressources matérielles ainsi que les performances

réseau. L'impact des différents paramètres réseau et de trafic, est également étudié. Nous avons implémenté les trois réseaux pour une performance maximale. Ainsi, les topologies DRAFT et fat-tree sont toutes deux basées sur des arbres binaires complets (quel que soit le nombre d'EC connectés). De même, tout mesh implémenté présente une structure carrée quel que soit le nombre d'EC connectés. La topologie mesh est générée via ATLAS [57] tandis que les topologies fat-tree et DRAFT sont générées par DRAGOON. Les outils d'implémentation (ISE 9.2i) et de simulation (ModelSim 9.5c) sont les mêmes pour les trois topologies. Les réseaux DRAFT et fat-tree sont générés avec la structure Fat-Node. Ainsi, les trois réseaux sont implémentés avec la même architecture de routeur et sans *Virtual Channels*. L'algorithme de routage est le seul composant qui diffère entre les trois topologies. Le réseau mesh utilise un routage de type *XY* [38] tandis que DRAFT et le fat-tree utilisent l'algorithme *Turn Back*. Tous les routeurs des trois topologies fonctionnent à 100MHz.

C.3-1 Ressources matérielles

DRAFT est défini comme un réseau qui supporte la reconfiguration dynamique partielle et minimise la consommation de ressources matérielles. La figure 3-11 présente les résultats d'implémentation en ressources et en nombre de liens pour les trois réseau. Chaque routeur est implémenté avec une taille de flit de 32 bits et une profondeur de buffer de 4 flits.



(a) Nbr de CLB=f(EC).

(b) Nbr de lien=f(EC).

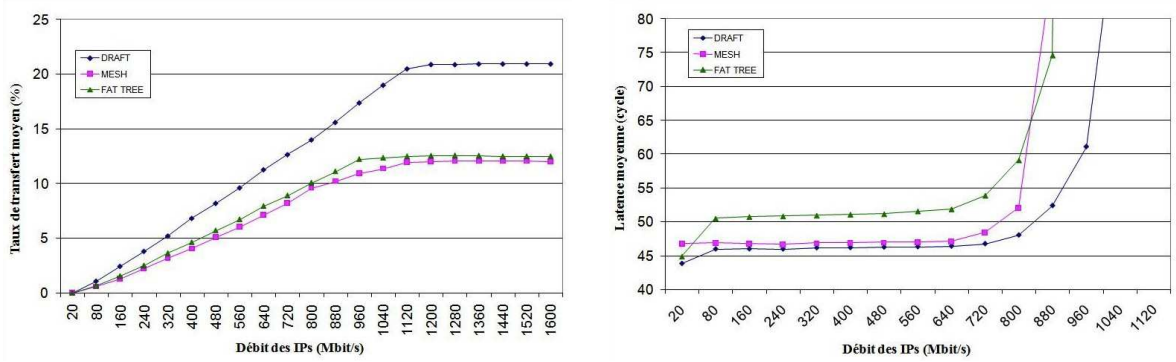
Figure 3-11 – **Consommation de ressources matérielles**. *Nombre de CLB et de liens utilisés dans un FPGA Xilinx Virtex 5 par les topologies DRAFT, fat-tree et Hermes en fonction du nombre d'EC connectés.*

De ces résultats d'implémentation, comme prévu, on voit qu'un fat-tree a besoin de plus de ressources matérielles et de liens de communication que les autres topologies. En utilisant la connexion des EC au sommet de DRAFT, ce réseau utilise moins de ressources logiques et moins de liens de communication qu'un mesh quand il y a moins de 16 EC connectés. Cependant, DRAFT dépasse les ressources utilisées par le mesh dans certaines plages (comme entre 17 et 25 EC connectés (Fig 3-11a)). Ce point est dû à l'hypothèse que DRAFT est implémenté comme un arbre binaire complet et le mesh comme une matrice carrée. Ainsi, DRAFT nécessite 32

routeurs pour connecter entre 17 et 32 EC. Le mesh lui utilise 25 routeurs pour connecter entre 17 et 25 EC, puis 36 routeurs pour connecter de 26 à 32 EC. Le nombre de fils de routage est un facteur limitant dans les FPGA. Le fait que DRAFT demande moins de liens de communication (Fig 3-11b) qu'un mesh est donc important.

C.3-2 Performances réseau

Habituellement, les latences et les taux d'utilisation (*throughput*) sont présentés en fonction du taux d'injection. Cependant cette notion diffère entre un mesh et un arbre. Ainsi, pour une comparaison équitable vis à vis de la topologie mesh, la figure 3-12 présente une comparaison des latences et taux d'utilisation en fonction du débit des EC. Les trois topologies sont simulées en connectant 8 EC avec une taille de flit de 32 bits et une profondeur de buffer de 4 flits. La fréquence des EC est de 100MHz et les données sont émises avec une répartition *uniforme* des sources et des destinations.



(a) Taux d'utilisation.

(b) Latence moyenne.

Figure 3-12 – **Performances réseau**. *Comparaison du taux d'utilisation moyen et des latences moyennes en fonction du débit des EC pour un trafic uniforme.*

De ces résultats, il apparaît que DRAFT bénéficie d'une latence moyenne plus faible tout en supportant des débits plus élevés que les autres réseaux. Cette faible latence est due à la réduction du nombre de routeurs, et donc de la longueur des chemins de communication. DRAFT supporte aussi un taux d'utilisation plus élevé quand on considère le débit des EC. Cela démontre une meilleure utilisation des ressources de routage que pour les topologies mesh et fat-tree.

C.3-3 Types de trafic de données

Dans cette partie, l'influence du trafic de données est présentée. Chaque réseau est conçu pour connecter 8 EC avec une taille de flit de 32 bits et une profondeur de buffers de 4 flits. Les trois types de trafic sont étudiés, et les données sont émises selon les distributions *uniformes*, *normales*, et *Pareto on/off*. Les résultats sont présentés dans le tableau 3-5. A partir de ces

résultats, DRAFT supporte mieux les différents types de trafic que les autres réseaux, même à débit maximum. Dans le cadre d'applications utilisant la reconfiguration dynamique partielle (où le trafic de données ne peut pas être prédit), il est intéressant de noter que notre réseau supporte les différentes distributions avec de meilleures performances que les autres topologies.

Type de trafic	DRAFT	fat-tree	mesh
uniforme (800 Mbit/s)	48,27	58,22	51,91
normale (800 Mbit/s)	43,40	56,71	49,21
Pareto (800 Mbit/s)	33,11	42,42	38,40
uniforme (400 Mbit/s)	46,20	51,07	46,87
normale (400 Mbit/s)	35,58	43,28	38,65
Pareto (400 Mbit/s)	31,43	39,83	35,64

Tableau 3-5 – **Influence des différents trafics de données sur les latences moyennes (en cycles d'horloge)**. *Mesures avec un débit maximal et un débit moyen.*

D Conclusion et perspectives

Nous avons, dans un premier temps, étudié l'apport de la logique multi-valuée par rapport à la couche physique des NoC. Nous avons mis au point des paires d'encodeurs/décodeurs rapides et basse consommation en logique MVL pour les interconnexions intégrées sur silicium. L'inconvénient majeur de cette approche réside, encore actuellement, dans la technologie d'implémentation, mais nul doute que l'on sera capable un jour de l'intégrer. La logique MVL permettra alors un vrai gain pour l'interconnexion comme il a été montré. Le deuxième inconvénient est que les technologies MVL augmentent la sensibilité au bruit en diminuant les échelles de tension. Deux approches peuvent alors être envisagées : soit en tirant partie du fait qu'il y a moins de fils dans ces technologies, soit en étudiant des codes CAC (*Crosstalk* Avoiding Code) spécifiques à ces technologies. C'est cette deuxième piste que nous privilégions.

Le travail sur le codage a montré l'efficacité de nos implémentations. Nous travaillons actuellement sur l'amélioration de la deuxième technique par un choix approprié du code de Berger. En effet, actuellement seul le nombre de 1 ou de 0 dans la donnée est utilisé pour ce choix. Nous avons montré qu'il est possible d'améliorer les performances du codage en choisissant le code de Berger afin de garder minimal le nombre de 1 dans la donnée finale (i.e. incluant le code lui même). Les premiers résultats montrent une diminution du nombre d'erreurs résiduelles de 23.5% pour un surcoût en surface nul et les estimations de consommation sont en cours. D'autres bases d'encodage sont aussi à l'étude, notamment l'utilisation du système de numération de Fibonaci, qui a montré des performances intéressantes en termes de consommation et d'efficacité d'implémentation.

Une revue sur les différents problèmes des NoC vis-à-vis de la reconfiguration dynamique nous a

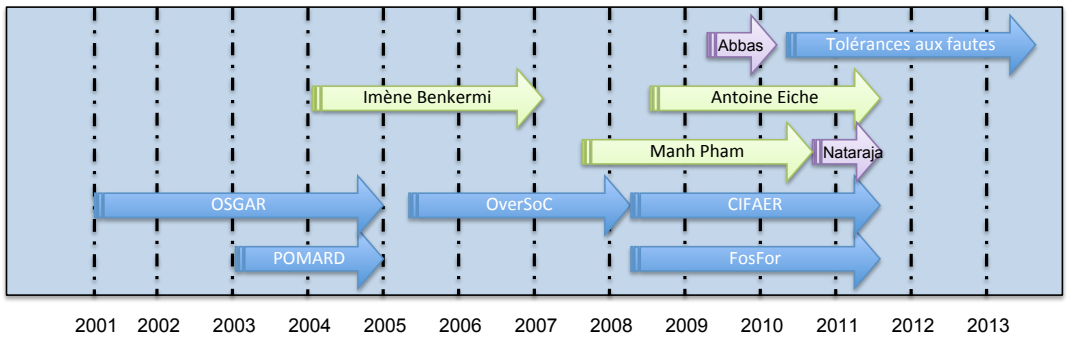
amené au réseau d'interconnexion flexible DRAFT. L'évolution actuelle du réseau implémente une architecture de routeur composée exclusivement de fils. La partie contrôle et routage des paquets est alors assurée par reconfiguration dynamique desdits routeurs. L'inconvénient majeur de cette approche réside dans les outils actuels qui nous obligent à générer l'ensemble des bitstreams pour chaque routeur (24 par routeur dans DRAFT). Une des solutions serait d'avoir des bitstreams "relogeables", permettant de ramener à 24 le nombre de bitstreams nécessaires pour tout le réseau.

L'évolution naturelle de mes travaux dans ce domaine est alors de définir une infrastructure de communication (de type NoC) qui supporte une adaptation dynamique de ses propres caractéristiques en fonction du contexte d'exécution d'un ensemble d'applications. Un des challenges de l'étude réside dans la variabilité de la localisation des tâches physiquement implémentées dans le RSoC à un instant donné et donc des communications qui les relient. Cette localisation aura un impact important sur les accès aux ressources partagées par exemple. L'adaptation dynamique du réseau de communication suppose, d'une part, la gestion dynamique des protocoles et des supports physiques de communication et, d'autre part, l'élaboration de métriques d'aide à la décision quant au choix raisonné (vitesse, consommation d'énergie, rapidité d'installation) de la meilleure stratégie de routage d'informations. De manière à être efficaces, les choix d'adaptations se feront de manière distribuée afin d'adapter au mieux l'ensemble des caractéristiques du réseau.

Chapitre

4 Gestion dynamique et efficace des ARD

Résumé : L'utilisation de plates-formes reconfigurables dynamiquement et les applications nécessitant de supporter des variabilités dans leur exécution ont un fort impact sur la gestion du système. Ainsi, l'utilisation d'un système d'exploitation devient nécessaire. Je présente dans ce chapitre mes travaux autour de la gestion efficace d'architectures reconfigurables. L'efficacité s'entend ici selon une vue optimisation de performances et simplification de programmation, mais aussi selon la vue gestion pour la tolérance aux fautes de ces architectures. La vue synthétique et chronologique de mes travaux et de mes encadrements dans ce domaine est représentée ci-dessous.



Sommaire

A Introduction

Aujourd'hui, la complexité algorithmique tend à s'accroître dans de nombreux domaines tels que le traitement d'images ou de signaux, le contrôle, ou encore les applications réseaux. Parallèlement, les applications embarquées requièrent de plus en plus de puissance de calcul afin de respecter des contraintes temps réel. Les concepteurs de circuits s'orientent alors vers des solutions parallèles et hétérogènes (logicielles et matérielles). Dans un souci de flexibilité, et comme nous l'avons vu, certains blocs matériels peuvent être reconfigurés dynamiquement au sein du circuit. Cette flexibilité permet, entre autres, de pouvoir s'adapter efficacement à un contexte applicatif de plus en plus dynamique. En contrepartie, l'utilisation de ces ARD pose néanmoins un problème en termes de contrôle et de gestion.

Aussi, il devient de plus en plus commun d'envisager l'utilisation de systèmes d'exploitation temps réel (RTOS – Real Time Operating Systems) dont le but est de fournir des services permettant, par exemple, de gérer les communications [82], l'exécution des tâches [24], leur séquencement [75]. Dans ce contexte, les traitements chargés dynamiquement dans les ARD sont vus comme des tâches matérielles pouvant communiquer avec les autres éléments de l'application. Concevoir un OS adapté est complexe et nécessite la prise en compte de l'architecture et de l'OS lui même afin d'offrir de réelles opportunités de développement. Dans le projet OveRSoC, un outil d'exploration a été développé. Cet outil permet d'aider le concepteur dans la description de sa plate-forme de manière à pouvoir explorer des choix critiques de conception portant notamment sur le système d'exploitation (distribution des services, services dédiés, etc.). Cette description est réalisée à haut niveau d'abstraction de manière à pouvoir valider les choix le plus tôt possible dans le flot de conception. Bien qu'ayant un grand intérêt en termes de simulation, OveRSoC ne permet pas l'implémentation du système d'exploitation sur une plate-forme réelle basée sur des circuits reconfigurables. Dans cette optique, nous étudions l'implémentation d'un OS spécifique dans le cadre du projet FosFor. L'originalité de l'approche vient de l'implémentation matérielle de services de l'OS afin de répondre aux contraintes temporelles des applications actuelles.

L'utilisation de technologies de gravure extrêmement fines permet l'intégration de systèmes de plus en plus complexes mais augmente dans le même temps les phénomènes de variabilité technologique et la sensibilité des circuits aux rayonnements électromagnétiques. Ainsi des fautes permanentes dues aux phénomènes de vieillissements et d'usures [59] apparaissent au cours de la vie du circuit. De plus, des fautes transitoires principalement dues à la radioactivité ambiante doivent aussi être supportées, notamment dans les applications critiques comme en aéronautique, dans le spatial ou encore l'automobile. La problématique est alors que ces fautes affectent particulièrement les points mémoires et notamment les mémoires de configuration des FPGA actuels [9]. Il est donc nécessaire d'intégrer des méthodes de gestion des erreurs dans les architectures reconfigurables dynamiquement. Nous avons développé deux techniques permettant d'utiliser la reconfiguration dynamique afin de supporter la tolérance aux fautes avec un faible surcoût matériel.

B Système d'exploitation pour le reconfigurable

Contexte
Cette partie porte sur des travaux qui ont été initiés dans le thème "système d'exploitation" de l'EPML POMARD et qui s'intègrent en partie dans le projet OveRSoC. Je travaille ici à la spécification et à l'implémentation d'un OS spécifique pour ARD.

La conception d'un RSoC nécessite de supporter l'hétérogénéité des composants et des logiciels constitutifs du futur système, mais doit aussi permettre la mise en œuvre de techniques de gestion adaptées à ces plates-formes. Nous définissons alors le SoC reconfigurable comme l'ensemble architecture/RTOS. De nombreux travaux s'intéressent à la modélisation et à la définition de systèmes d'exploitation [4][28] [5] [54]. Néanmoins, aucun ne permet de prendre en compte, dans la phase de conception, le développement de l'OS lui même.

B.1 Impact de la reconfiguration dynamique sur les services de l'OS

Le rôle de l'OS consiste à organiser l'ensemble des traitements sur la plate-forme RSoC. La présence d'ARD dans le système modifie assez profondément plusieurs services de l'OS. En effet, l'ARD fait apparaître la notion de tâche "hardware", représentant un calcul implémenté sous la forme d'un bloc dédié. La forme exécutable de ce type de tâche est un bitstream. La figure 4-1 résume les propriétés intrinsèques d'une architecture reconfigurable dynamiquement qui impactent les services de l'OS.

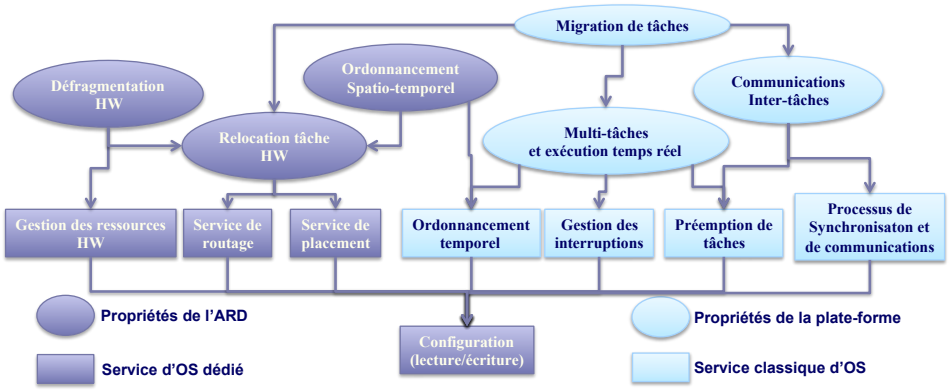


Figure 4-1 – **Services d'un OS supportant la gestion d'une ARD**. *En plus des services classiques d'un OS logiciel, des services de gestion particuliers sont requis. L'OS doit être capable de gérer des types de tâche différents (implémentation logicielle ou matérielle) et les services doivent supporter l'aspect dynamique de l'architecture.*

En plus des propriétés classiques que l'on attend, un RTOS gérant une ARD nécessite la *gestion des tâches* matérielles, et ce de manière dynamique. L'OS peut être amené à gérer plusieurs

instances d'une même tâche. En effet, certaines tâches peuvent être instanciées au travers de tâches logicielles et/ou matérielles. Le choix du type pour l'exécution d'un traitement est réalisé dynamiquement par l'OS. Le fait d'avoir une ARD permet aussi d'envisager de dupliquer des machines virtuelles (processeur soft-core) sur l'ARD et ainsi d'envisager l'équilibrage de charge entre les cibles logicielles et matérielles. Dans ce cas, un service particulier est requis car il apparaît ici la notion de phase de configuration optionnelle. En effet, si une tâche logicielle doit être affectée à une machine virtuelle, la phase de configuration de cette même machine virtuelle peut être optionnelle, si elle est déjà présente dans l'ARD.

Quelle que soit la stratégie choisie lors de l'instanciation d'une tâche matérielle, il faudra alors décider d'un placement de la tâche dans la matrice de l'ARD. Afin de répondre aux besoins de performances et d'optimisation, ce dernier doit être réalisé de manière dynamique et flexible. L'ordonnancement *spatio-temporel*, doit alors gérer l'exécution des tâches non seulement dans le domaine temporel, mais aussi dans le domaine spatial. Ce nouveau type d'ordonnanceurs doit alors gérer les différents types de tâches en fonction de contraintes de temps réel, mais aussi de disponibilité de ressources. Ce service de placement peut aussi offrir un service plus évolué si l'on souhaite assurer la répartition optimale des charges de calcul au sein du RSoC. Pour remplir cet objectif, les tâches ainsi que leurs supports d'exécution doivent supporter une éventuelle préemption par l'OS.

La gestion des ressources est aussi un point clé pour la gestion d'une ARD. Une tâche matérielle doit être implémentée de manière la plus efficace possible, donc en utilisant les ressources adéquates. La notion de ressources dans le modèle de RSoC est une notion dynamique car elle évolue au fil des reconfigurations effectuées par l'OS. Cette propriété a un impact sur le placement des tâches, les architectures comportant différentes ressources hétérogènes (blocs de logique, mémoires, multiplieurs câblés, ...). L'allocation dynamique de tâches au sein de l'ARD peut entrainer une fragmentation des ressources (de manière similaire à une mémoire dans laquelle une succession d'allocations et de désallocations sont effectuées). La propriété de *défragmentation* permet alors de compacter les tâches au sein de l'architecture afin de récupérer la forme rectangulaire la plus grande possible. La gestion de la mémoire est aussi un réel enjeu si l'on souhaite conserver les performances et la flexibilité qu'apporte l'utilisation d'ARD.

Les propriétés précédentes s'appuient sur la propriété de *relocation de tâche*, qui permet de déplacer une tâche matérielle au sein de l'ARD. Cette propriété est nécessaire pour l'implémentation de la migration de tâche de matériel vers matériel. L'*InterMigration*, qui permet de migrer d'une cible, matérielle ou logicielle, vers une autre cible d'un type différent doit être réalisée en fonction de critères spécifiques comme le temps d'exécution global. Cela suppose l'existence de deux instances différentes (l'une logicielle, l'autre matérielle) d'une même tâche. Cela suppose aussi qu'il existe, à un instant donné, une équivalence de représentation dans le cycle d'exécution de la tâche permettant un échange entre ses deux représentations.

Toutes ces propriétés avancées peuvent permettre une optimisation des temps d'exécution uniquement si la propriété de communication *inter-tâches* supporte la flexibilité requise. Le fait

de pouvoir allouer dynamiquement des tâches, ou de les déplacer, nécessite de supporter une continuité dans les communications entre les tâches ou les ressources. Ce service doit donc assurer la cohérence des échanges de données entre des tâches logicielles et matérielles.

Les propriétés offertes par la reconfiguration dynamique nécessitent donc de revisiter certains services d'OS (figure 4-1). Des services spécifiques comme la gestion des ressources matérielles, la gestion de la reconfiguration, le service de placement et de routage sont nécessaires pour l'implémentation des propriétés précédentes. Cependant, d'autres services nécessitent d'être adaptés afin de prendre en compte les spécificités d'un RSoC (ordonnancement, préemption et migration de tâches, communication). Cette liste non exhaustive est, à mon sens, minimale. La conception d'une plate-forme reconfigurable incluant un OS spécifique est alors une tâche complexe nécessitant la mise en œuvre d'une méthodologie adaptée.

B.2 Modélisation

Le projet OveRSoC propose une méthodologie permettant de modéliser les services d'un système d'exploitation destiné à gérer un RSoC. Les entrées du flot d'exploration consistent à spécifier non seulement l'application, mais également la plate-forme sur laquelle cette application va être exécutée. La plate-forme minimale est composée de trois composants (Fig 4-2) : le système d'exploitation, une unité reconfigurable et un processeur.

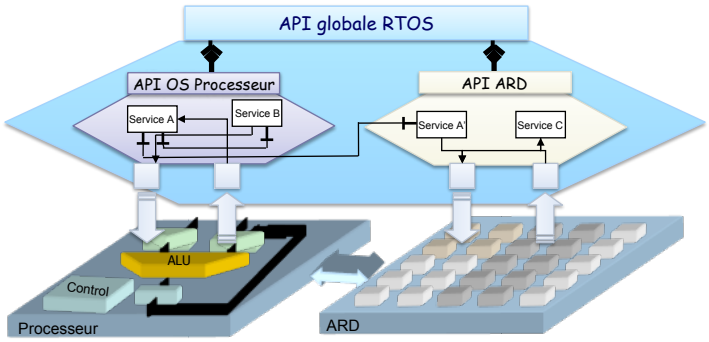


Figure 4-2 – **Structure de la plate-forme reconfigurable OverSoC**. *Elle est constituée au minimum d'un processeur généraliste, d'une ARD et d'un OS. L'OS est distribué au travers de la plate-forme afin de mieux répondre aux spécificités de chaque élément.*

L'interface de programmation (API – Application Programming Interface) de l'OS est constituée de la composition des services déployés sur les différentes unités d'exécution. L'OS est donc distribué sur la plate-forme afin d'optimiser et d'adapter ses services à chaque type de ressource. L'objectif est alors de proposer une exploration des services de l'OS et de leur implémentation. La méthode est basée sur des modèles abstraits et modulaires des différents éléments de la plate-forme. Dans le projet, nous avons développé le modèle multi-niveau (figure 4-3) de l'ARD.

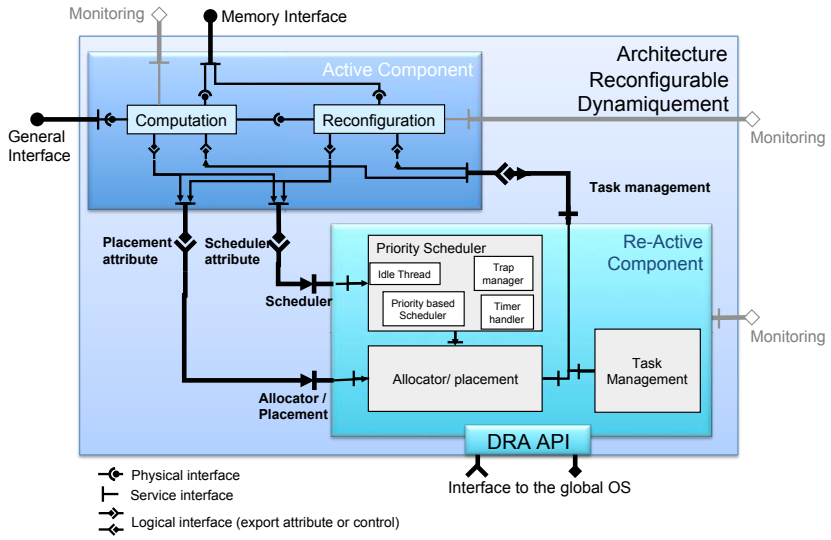


Figure 4-3 – **Modèle de l'architecture reconfigurable dynamiquement**. *La partie "Active component " représente les éléments physiques de l'architecture, tandis que la partie "Re-Active component" intègre les aspects OS et gestion de l'architecture.*

Le premier élément de ce modèle est le *Active component*, il représente les éléments matériels de l'architecture ainsi que les contraintes physiques, comme par exemple le nombre et le type de ressources disponibles. Ce composant composite est lui même constitué de deux composants indépendants. Le composant *computation* regroupe les éléments constitutifs de l'architecture (blocs logique reconfigurable, mémoires, multiplieurs câblé, ...), alors que le composant *reconfiguration* encapsule les éléments permettant la reconfiguration physique du composant (par exemple l'ICAP pour les FPGA Xilinx). Le *Re-Active component* représente le comportement dynamique de l'architecture. La figure 4-3 présente une instance particulière d'ARD incluant un ordonnanceur basé sur des priorités, et intègre les services d'allocation et de gestion des tâches. Ce composant composite peut être étendu avec d'autres services ou propriétés en fonction des besoins. Tous les services dédiés de l'OS sont inclus dans ce composant. Dans le modèle, l'interface externe du *Re-Active component* constitue l'API de l'OS de gestion de l'ARD, alors que l'interface du *Active component* représente les interfaces physiques du circuit (avec un circuit mémoire par exemple). La définition des composants *Active* et *Re-Active* donne une encapsulation, une configuration et une composition du système.

Afin de permettre l'exploration de la plate-forme, ainsi que de vérifier le comportement des différents composants, des points de test et de mesures (monitoring sur la figure 4-3) ont été prévus. Dans le modèle, les interfaces entre les composants représentent les interactions entre la partie contrôle et la partie calculatoire. D'un côté, les attributs nécessaires à la gestion de l'architecture sont exportés vers le *Re-Active component*, tandis que les interfaces internes des services de l'OS permettent le contrôle de la reconfiguration. La définition des composants (offrant/nécessitant un service) est indépendante et permet une exploration des services de l'OS et de la plate-forme. Chaque composant est constitué par des méta-objets qui peuvent

être définis indépendamment et décrits avec différents niveaux de représentation.

La définition de ce modèle est formalisée par un diagramme de classes UML tandis que l'implémentation de chaque composant est réalisée en SystemC. Afin de simplifier le processus, nous avons défini trois niveaux de spécification, chacun ajoutant des détails dans l'implémentation. Toute cette méthodologie est supportée par l'outil DOGME [65] qui a été développé de manière à rendre l'utilisation de la méthodologie plus conviviale. Cet outil constitue un greffon indépendant de la plate-forme *eclipse*. Le rôle de l'outil consiste à supporter la spécification schématique de l'ensemble de la plate-forme. La description est basée sur l'utilisation de composants prédéfinis dans une bibliothèque dont l'utilisateur peut préciser les différentes interconnexions (Fig 4-4). La plate-forme est alors générée en SystemC à des fins d'évaluations de performances.

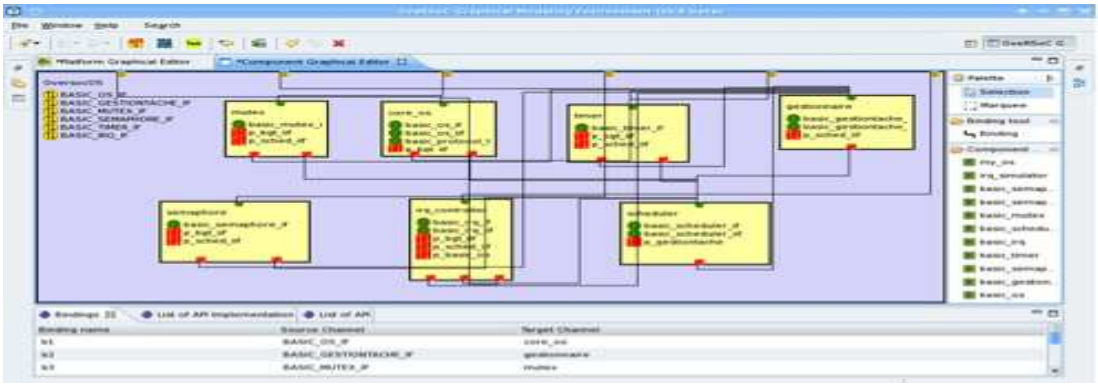


Figure 4-4 – **Outil Dogme**. *Basé sur un greffon eclipse, cet outil supporte la méthodologie OverSoC. Après la définition d'une architecture, les éléments constitutifs de la future plate-forme sont assemblés afin de générer un code SystemC simulable.*

B.3 Conception d'un service de placement-ordonnancement

Nous nous plaçons dans le cadre d'un FPGA constitué uniquement de CLB. Dans le premier niveau du modèle proposé ci-dessus, l'ARD (i.e. le *Active component*) n'est représentée que par un nombre de ressources maximal. L'objet *reconfiguration* est implicite et n'est pas contrôlé dans cet exemple. Le service de placement est alors décrit dans le *Re-Active component*. Cette première description permet de fixer l'API de ce futur service. Le listing 4-1 présente le code SystemC de ce placeur initial qui ne vérifie que la disponibilité des ressources.

Cette première évaluation permet, par exemple, de dimensionner la surface maximale du circuit nécessaire. Un autre intérêt de cette simulation est de définir les interfaces physiques de l'ARD sans avoir à implémenter trop de détails. Dans ce contexte, l'exécution d'une application constituée de trois tâches (T_1 une tâche logicielle et T_2 et T_3 deux tâches matérielles) donne la trace d'exécution présentée sur le listing 4-2. On voit que les deux tâches matérielles sont acceptées (lignes 10 et 11).

Une fois les API bien définies et l'ensemble des services requis décrit à un haut niveau d'abstraction, le concepteur peut commencer le raffinement de sa description. Dans le deuxième niveau,

```
1  int simple_ard_placer :: place_tacheARD (Task_mod* TacheHW )
2  {
3     if(Local_DRA_Resources >= TacheHW->get_resources())   {
4        Local_DRA_Resources -= TacheHW->get.resources();
5        return OVERSOC_OK;
6     }
7     else  {
8        return -1;
9     }}
```

Listing 4-1 – ***Description SystemC du placeur initial*** *Ce placeur ne fait que vérifier si l'ARD a suffisamment de ressources pour accepter la tâche matérielle.*

```
1                SystemC 2.2.0 --- Sep 12 2009 13:42:31
2          Copyright (c) 1996-2006 by all Contributors
3  ==> Degug at 0 s from <OS> : Create task T1 is done
4  ==========>at time 0 s Entering scheduler...1 tasks, , Tasks list
5                type=SOFT    pid=0; pri=1; name=T1; state=ready
6  ==========>at time 0 s Exiting scheduler...
7  Finishing task with PID: 0 at time: 110 ns
8  ==> Degug at 110 ns from <T1> : Create Task T2
9  ==> Degug at 110 ns from <T1> : Create Task T3
10 ==> Degug at 200 ns from <DRA> : Load Task T2 : OK
11 ==> Degug at 300 ns from <DRA > : Load Task T3 : OK
12 ==========>at time 1 us Entering scheduler...2 tasks, Tasks list
13               type=HARD    pid=1; pri=1; name=T2; state=running
14               type=HARD    pid=2; pri=1; name=T3; state= running
```

Listing 4-2 – ***Trace d'exécution du niveau 1 du service de placement.*** *Les trois tâches sont créées et l'ARD accepte les tâches matérielles car le nombre de ressources est suffisant.*

le *Active component* est décrit de manière plus précise, comme étant par exemple une matrice de CLB. Dans ce contexte, la seule information de nombre de ressources n'est plus suffisante et la forme des tâches rentre alors en ligne de compte. L'exécution de l'exemple précédent pour lequel nous avons spécifié des surfaces et des tailles pour chaque tâche donne la trace d'exécution du listing 4-3. Dans ce cas l'ARD accepte la tâche T_2 mais rejette la tâche T_3 (ligne 8) pour un problème d'espace. Des services plus évolués peuvent être rajoutés ou définis à ce niveau. Par exemple, le service d'allocation de tâches pourrait instancier la tâche T_3 sous sa forme logicielle pour optimiser les temps d'exécution.

```
1                SystemC 2.2.0 --- Sep 12 2009 13:44:18
2          Copyright (c) 1996-2006 by all Contributors
3  ==> Degug at 0 s from <OS> : Create task T1 is done
4  ?
5  ==> Degug at 110 ns from <T1> : Create Task T2
6  ==> Degug at 110 ns from <T1> : Create Task T3
7  ==> Degug at 200 ns from <DRA> : Load Task T2 : OK
8  ==> Degug at 300 ns from <DRA > : Load Task T3 : : Not enough space
9  ==========>at time 1 us Entering scheduler...1 tasks, Tasks list
10               type=HARD    pid=1; pri=1; name=T2; state=running
```

Listing 4-3 – ***Trace d'exécution de niveau 2 du service de placement.*** *Les tâches T_1 et T_2 sont créer, mais pas la tâche T_3 par manque de surface.*

Le dernier niveau de description prend en compte l'ensemble des spécificités de l'architecture et des services associés. Par exemple, l'hétérogénéité des ressources des ARD peut être considérée.

Dans le cadre de nos travaux, nous avons alors modélisé et implémenté des algorithmes de placement de tâches au sein de la zone reconfigurable.

La majorité des algorithmes de placement utilise une gestion des zones libres (MER – Maximum Empty Rectangle) [24]. Lorsqu'une tâche arrive, elle est placée dans un rectangle choisi selon une stratégie (*First_Fit*, *Best_Fit*, ...) et un nouvel ensemble de MER est calculé. Le listing 4-4 présente le code SystemC de l'algorithme *First_Fit* qui choisit le premier rectangle vide qui a une taille suffisante pour accueillir la nouvelle tâche. Il est à noter que la mise à jour et la création de la liste des MER sont indépendantes de l'algorithme de placement en lui même.

```
1  //Calcul du placement de tâche selon l'algorithme First_Fit
2  void simple_ard_placer :: First_Fit(Task_mod* TacheHW)
3  { std::list<MER*>::iterator tl1; // L'itérateur  sur la liste des tâches
4     bool FindPos=false;
5     for (tl1=MER_list.begin(); tl1!=MER_list.end(); tl1++) {
6        if ((*tl1)->get_width()>=TacheHW->get_width() and (*tl1)->get_height()>=TacheHW->
             get_height()){
7           FindPos=true;
8           updateMatrix((*tl1)->get_x0(),(*tl1)->get_y0(),TacheHW->get_width(),TacheHW->
                get_height(),TacheHW->get_task_ID());
9           break;
10       }
11    }
12    if (!FindPos) {
13       cout << "WARNING: Task " <<TacheHW->get_name() << "cannot be placed on matrix" <<endl;
14    }
15 }
```

Listing 4-4 – ***Description SystemC de l'algorithme* First_Fit** *Ce placeur cherche dans la liste des MER disponibles si un rectangle est suffisamment grand pour accueillir la nouvelle tâche. Dès qu'il en trouve un, il affecte la tâche dans cet espace et recalcule la liste des MER.*

L'algorithme *First_Fit* n'est pas optimal puisque le choix du rectangle pour accueillir la tâche n'est pas celui qui minimise la fragmentation de la zone reconfigurable. Afin d'obtenir de meilleurs résultats, nous étudions à l'heure actuelle des algorithmes de placement plus évolués tel que le *Best_Fit*, le *stuffing* ou le *horizon* [84]. Ces méthodes plus évoluées permettent un placement plus optimisé mais requièrent aussi des temps de calcul plus long.

La méthodologie proposée permet donc de réaliser l'exploration de l'espace des solutions pour la conception de services d'OS pour une ARD de manière itérative et en se concentrant sur certaines parties du système.

Nos travaux précédents sur l'utilisation des réseaux de neurones pour l'ordonnancement sur architectures hétérogènes (Thèse d'Imène Benkermi [11]) nous ont amené à étudier un service de placement lui aussi basé sur les réseaux de neurones. La mise en commun de ces propositions constituera alors un ordonnancement spatio-temporel pour RSoC exécutable dynamiquement. Le temps de convergence des réseaux de neurones proposés, allié à une implémentation matérielle efficace, permettent d'envisager un ordonnancement dynamique en-ligne. Les modèles proposés ont montré [ECPS10] que notre solution est très efficace par rapport aux algorithmes de placement classiques étudiés ci-dessus.

C Système d'exploitation pour MPSoC reconfigurable

Contexte
Ces travaux sont menés entièrement dans le projet FosFor. Il s'agit de proposer un OS flexible gérant une plate-forme multi-processeur intégrant un accélérateur reconfigurable.

Dans ce projet, nous nous appuyons sur des mécanismes de virtualisation des services supportés par un *middleware*. Cette virtualisation est nécessaire pour que les tâches de l'application s'exécutent et communiquent sans connaissance a priori de leur affectation à une unité de traitement.

C.1 Architecture FosFor

Dans le projet FOSFOR, la plate-forme est de type multiprocesseurs on chip (MPSoC), le terme « processeur » s'étendant alors à tout type d'unités d'exécution (processeurs généralistes, architectures reconfigurables de type FPGA, processeurs reconfigurables). Il existe donc deux domaines d'exécution (figure 4-5). Le premier que l'on peut qualifier de « software » est composé d'un ensemble (au moins un) de processeurs généralistes (GPP – General Purpose Processor), d'une mémoire partagée au sens logiciel du terme et de périphériques d'entrées/sorties. Un GPP peut supporter l'exécution de services de l'OS (*SoftOS*) et peut prendre en charge l'exécution de *threads* logiciels (*SoftwareTask*). Tous les GPPs ne sont pas forcément homogènes en termes de type et de nombre de services offerts.

Le deuxième domaine est dit « hardware », il est constitué d'un ensemble (non vide) de zones reconfigurables dynamiquement (RR – Région Reconfigurable). Une RR est une zone qui peut prendre en charge l'exécution d'un ensemble de tâches matérielles (*HardwareTask*), simultanément ou séquentiellement. Comme pour un GPP, une RR peut supporter l'exécution de services de l'OS (*HardOS*). Ces zones peuvent être grain fin (FPGA) ou de grain plus épais (processeur reconfigurable). Dans le projet FosFor, nous nous focalisons sur une architecture grain fin de type FPGA. Dans ce domaine, nous distinguons des zones reconfigurables (RZ – Reconfigurable Zone, équivalent des PRR Xilinx) qui sont les zones physiques où seront mappées les tâches de l'application ; des mémoires partagées qui sont vues dans l'espace adressable du système par le *middleware* ; et un wrapper (NoC IF) vers le domaine logiciel de la plate-forme. L'ensemble est interconnecté par le NoC DRAFT (§III.C, page 62) et un bus de contrôle dédié au *middleware*.

Les domaines communiquent grâce à des canaux virtuels de communication. A ce niveau de description, aucune précision n'est apportée quant aux supports physiques de ces canaux. Nous en identifions trois types différents :
– canal de contrôle : il permet la communication entre les services distribués de l'OS,
– canal de données : c'est lui qui véhicule les données,
– canal de configuration : transfère les configurations des tâches logicielles (code binaire) et matérielles (bitstreams) entre la mémoire de configurations et les cibles d'exécution.

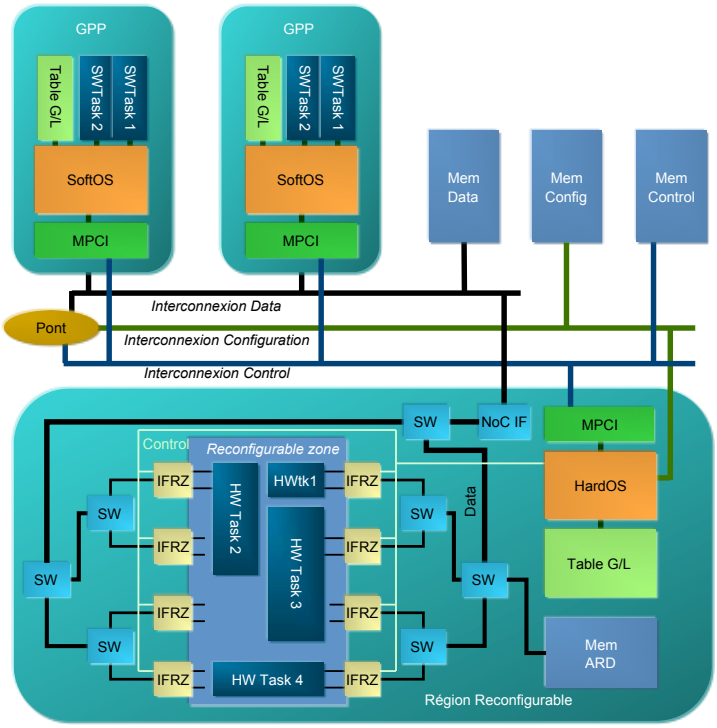


Figure 4-5 – **Architecture Fosfor**. *Construite autour de processeurs généralistes et d'au moins une zone reconfigurable dynamiquement, elle est gérée par un OS multiprocesseurs distribué.*

Notons la présence d'un pont entre les différents médias de communication pour permettre des transferts entre les différentes mémoires. Chaque processeur dispose de sa propre mémoire locale. Cette mémoire stocke les données locales et, dans le cas des GPP, renferme le code de la tâche. Il n'est donc pas fait de distinction entre mémoire de données et mémoire de programmes. Le partage de données entre les tâches est rendu possible par une mémoire partagée connectée sur le canal de données. Les mémoires disponibles dans la zone reconfigurable (incluses dans l'espace d'adresse général) sont présentes au sommet du réseau DRAFT (figure 4-5).

Le modèle proposé repose sur une certaine régularité d'un point de vue de ses ressources de calcul, comme d'un point de vue de l'accès aux mémoires par le biais des supports de communications. Ainsi, chaque ressource peut accéder à des mémoires partagées, contenant les données et les programmes ou configurations des ressources d'exécution. De même, chaque ressource peut accéder à une mémoire de contrôle lui permettant de sauvegarder et de restituer son contexte d'exécution.

Cette structure donne toute liberté quant à la capacité d'implémenter n'importe quel service de l'OS sur n'importe quelle ressource d'exécution. Dans ce sens, une piste intéressante est une implémentation matérielle de services particulièrement complexes. La plate-forme doit donc permettre l'intégration des tâches de l'OS en plus des tâches applicatives et devra permettre un échange entre des services dont l'implémentation est hétérogène.

Parmi l'ensemble des OS existants, nous nous basons sur RTEMS [62]. C'est un système d'ex-

ploitation temps réel pour systèmes multiprocesseurs. Le point fort de RTEMS est que c'est un OS distribué. Pour cela, il intègre une couche spécifique appelée MPCI (MultiProcessor Communication Interface). Le principe du MPCI permet aux tâches s'exécutant sur un *node* de ne pas avoir à tenir compte de la localisation des objets auxquels elles font référence. En effet, l'utilisation d'un objet résidant sur un *node* distant s'effectue de la même manière que l'utilisation d'un objet local (sur le même *node*). Chaque *node* est informé lors de la création ou de la suppression d'un objet global, et maintient ainsi une table des objets globaux à jour (Table G/L figure 4-5). Lors de l'utilisation d'un tel objet, RTEMS détermine automatiquement le *node* sur lequel il réside à l'aide de la table globale, et utilise alors le MPCI pour transférer la requête au *node* correspondant. La réponse arrive par le même biais, et est redirigée vers la tâche émettrice qui s'était mise en attente. Le fonctionnement est donc totalement transparent pour la tâche. Dans le cadre du projet FosFor, nous avons donc adapté ce mécanisme à l'OS matériel qui gère la RR.

C.2 Service de communication intelligent

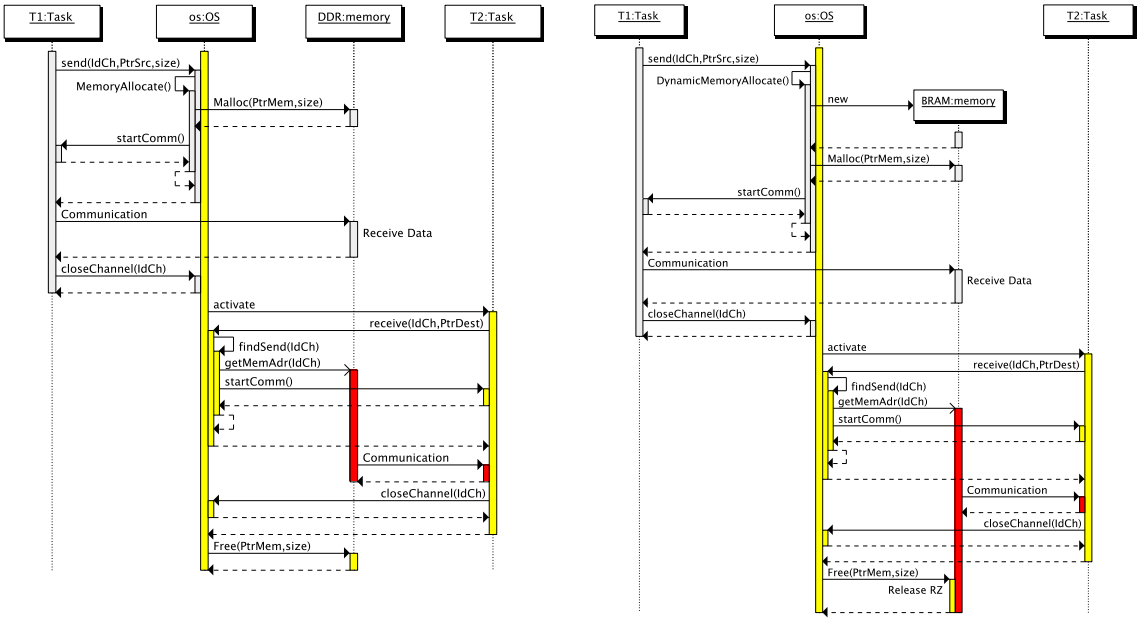
Plus que tout autre service, les communications sont fortement impactées par la distribution de l'OS et des tâches. Les objets de communications peuvent prendre plusieurs formes (file de messages, fichiers, mémoire partagée) et la distribution des tâches au sein d'une plate-forme multi-ressources hétérogènes pose le problème de la localisation du support physique de la communication.

Du point de vue matériel, nous avons défini la notion de tâche matérielle inspirée d'un thread matériel proposé dans ReconOS [54]. Une tâche hardware possède une interface dédiée avec le NoC, une interface avec l'OS matériel et une mémoire locale (au même titre que les GPP). Une tâche matérielle a donc la visibilité de l'espace d'adresse global mais possède aussi un espace local privé. Les différents services du *middleware* sont véhiculés par le bus de contrôle. Afin de virtualiser la communication, les tâches devront passer par l'ouverture d'un canal virtuel, service offert par le *middleware*. Nous considérons dans une première approche que les communications sont atomiques. Elles sont synchronisées par l'ouverture du canal virtuel. La tâche émettrice envoie ses données par l'intermédiaire d'une commande de type *send* et de son coté la tâche réceptrice attend ses données sur une opération *receive*. Une fois la communication terminée, le canal est refermé par les deux tâches.

Etant donné la plate-forme FosFor, le premier scénario d'échange concerne deux tâches logicielles. Dans ce cas, l'échange se fera via la couche logicielle MPCI et utilisera donc des mécanismes "classiques" des systèmes multiprocesseurs. Nous ne détaillerons donc pas ce cas dans la suite.

L'échange entre une tâche logicielle et une tâche matérielle se fera via une mémoire partagée, définissant une communication "avec copie". Dans la plate-forme FosFor, il existe deux types de mémoires (figure 4-5) : la mémoire globale (*data mem*) et les mémoires locales se trouvant

dans la RR (*BRAM* pour les circuits Xilinx) mais étant dans l'espace d'adressage du système. Ce type de transaction, représenté sur la figure 4-6a, correspond à un échange standard par passage de message. Après l'ouverture du canal virtuel (non représenté), le *middleware* s'occupe de récupérer une adresse à fournir aux tâches afin de procéder à l'échange. La différence majeure vient de la méthode de récupération, car dans le cas où une tâche matérielle (ou une BRAM) est impliquée dans la transaction, il faut alors gérer les adresses physiques du NoC permettant l'accès à l'une ou l'autre des mémoires. La tâche émettrice T1 fait un appel système *send* et attend, de la part du *middleware*, l'adresse du destinataire. Ici le *middleware* lui fournit une adresse qui correspond à un bloc mémoire. Lorsque la tâche T2 effectue son appel *receive*, alors le *middleware* doit déclencher l'équivalent d'un *send* dans le bloc mémoire (Memory). Le bloc mémoire doit donc disposer d'une interface réseau et d'une machine d'états système lui permettant de répondre aux sollicitations du *middleware*



(a) Echange par mémoire globale

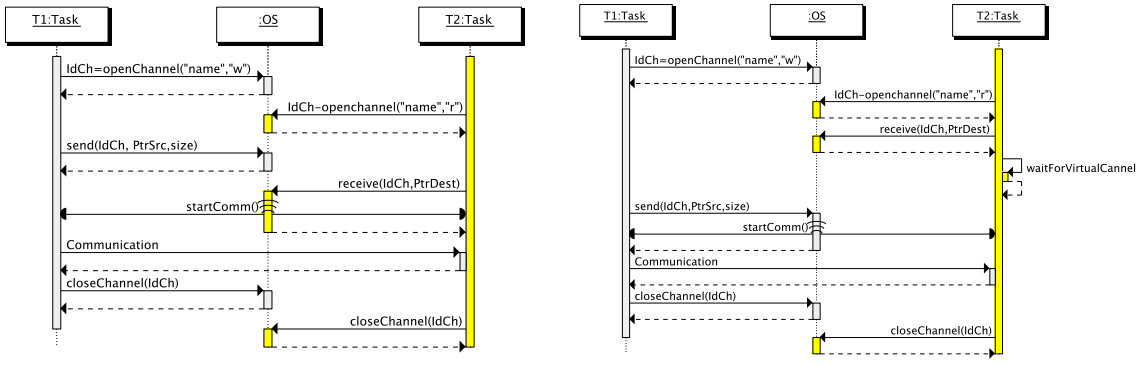
(b) Echange par mémoire dynamique

Figure 4-6 – **Synchronisation des appels de fonctions *send* et *receive* lors d'un échange de données "avec copie"**. *Les deux tâches ouvrent un canal de communication (non représenté) avant d'effectuer l'échange. Dans le cas de deux tâches matérielles, une création dynamique de mémoire peut avoir lieu dans une RZ.*

Cependant la présence de l'ARD permet d'envisager un mode de communication supplémentaire par la création dynamique de zones de mémorisation temporaires. En effet, dans le cas où il existe une RZ non utilisée par une tâche, il est possible d'utiliser ses ressources pour faire du stockage de données. Dans ce cas (figure 4-6b), le *middleware* décide de stocker un message dans une mémoire pour pallier à l'absence de la tâche réceptrice au moment de l'émission. La différence par rapport au cas précédent réside dans le choix de la mémoire pour effectuer ce stockage. On parle alors d'échange par mémoire dynamique. Le problème majeur est ici de

garder l'espace adressable consistant, car ces mémoires dynamiques ne sont pas dans l'espace d'adressage global. Dans ce scénario, la tâche T1 émet un appel *send*. Le *middleware* recherche la tâche destinatrice. Ne la trouvant pas, il décide de stocker le message au sein d'une mémoire allouée dynamiquement dans une zone reconfigurable. Le *middleware* configure donc la RZ en bloc mémoire et alloue la place nécessaire au sein de cette mémoire. Le *middleware* retourne alors l'adresse réseau du bloc mémoire alloué à la tâche T1. La suite de la communication est la même que précédemment, sauf à la fin de la communication, où le *middleware* désalloue la mémoire au sein de la zone reconfigurable. On notera que la communication avec copie permet le fait que les deux tâches impliquées dans la communication ne soient pas instanciées au même moment.

Dans le cas d'une communication entre deux tâches matérielles il est possible, en plus du mode "avec copie", de réaliser une communication "sans copie" au sens de la mémoire partagée car les tâches possèdent une mémoire locale. Dans ce cas de figure, le support de communication interne à la zone reconfigurable (le NoC) doit pouvoir supporter la description et la configuration d'une connexion directe. Les deux tâches doivent donc être présentes à l'instant où la communication est réalisée et cela jusqu'à la fin de l'échange. En effet, les tâches ne peuvent en aucun cas être interrompues tant que l'échange n'est pas terminé sous peine de perdre les données dans le cas d'une reconfiguration par exemple. Dans les exemples de la figure 4-7, l'échange s'effectue entre deux tâches matérielles (T1 et T2) configurées dans des zones reconfigurable (RZ) différentes. La tâche émettrice T1 crée le canal de communication en identifiant celui-ci par son nom, puis prépare les données à échanger dans sa mémoire. Elle effectue un appel système de type *send*. La tâche réceptrice ouvre le canal en lecture en l'identifiant par son nom, puis effectue un appel système de type *receive*. La communication est bloquante pour les tâches et la synchronisation (ou rendez-vous) est assurée par le *middleware*.



(a) Echange standard

(b) Appel bloquant

Figure 4-7 – **Synchronisation des appels de fonctions *send* et *receive* lors d'un échange de données "sans copie"**. *Les appels* send *et* receive *sont bloquants. Les deux tâches ouvrent le canal de communication avant d'effectuer l'échange. Le canal est identifié par son nom par les tâches (donc par le programmeur). Dans le deuxième cas, la tâche T2 est suspendue dans l'attente du* send *de la tâche T1.*

Dans le cas de la figure 4-7a, la tâche réalisant le *send* sera suspendue par le système tant que la tâche réceptrice n'aura pas effectuée son appel *receive*. A l'inverse, la tâche réalisant l'appel receive sera bloquée tant que la tâche émettrice n'aura pas exécuté l'appel système *send* (figure 4-7b).

Nous avons spécifié et formalisé ces scénarios d'échanges [DPCD10c] ; nous travaillons actuellement à leurs implémentations matérielles pour le contrôle du NoC DRAFT et dans le cadre de l'OS matériel du projet FosFor.

D Tolérances aux fautes

Contexte
Ce travail consacré à la prise en compte de la reconfiguration dynamique pour gérer la tolérance aux fautes est mené dans le cadre du projet CIFAER et de la thèse de H. Pham. Dans ces travaux, nous proposons de démontrer l'intérêt des technologies reconfigurables dynamiquement dans le domaine de la sûreté de fonctionnement.

Dans un système électronique, il existe deux types de fautes. L'erreur temporaire de nature aléatoire, et l'erreur permanente généralement due au vieillissement et à l'usure des composants. La conception matérielle doit prendre en considération ces deux types de fautes pour assurer un certain niveau de sûreté de fonctionnement. La reconfiguration dynamique offre alors de nouvelles stratégies, mais nécessite aussi de nouvelles approches pour prendre en compte ses spécificités.

Les dispositifs électroniques sont soumis aux rayonnements électromagnétique, comme les particules alpha et les neutrons provenant du cosmos. Les états des transistors peuvent alors changer, cet effet est appelé SEU (Single Event Upset). Les mémoires de technologie SRAM, notamment utilisées dans les FPGA, sont très sensibles à ce type de phénomènes [14]. La protection de la mémoire de configuration est alors primordiale. Les FPGA Xilinx, offrent la possibilité de relire le contenu de la mémoire de configuration par *Readback* [87]. Cette technique permet de vérifier que les données actuelles de configuration sont correctes, mais aussi de lire l'état courant de l'ensemble des éléments et de mémorisation du plan de calcul. Le *scrubbing* [12] consiste à reconfigurer périodiquement le FPGA à une fréquence supérieure à la fréquence d'apparition des fautes. C'est la technique actuellement la plus utilisée. Une dernière technique est le vote de CRC [18], l'idée est de calculer les CRC des zones mémoires correspondants à des modules dupliqués. Un vote est alors effectué sur ces signatures afin de détecter les fautes. On peut ainsi rétablir le système en copiant la mémoire de configuration du FPGA sans faute dans la mémoire du FPGA défaillant. Cette technique fonctionne actuellement sur des systèmes multi-FPGA et ne peut pas être appliquée au sein d'un même circuit.

Le *tiling* [42] est la seule technique qui permet de palier les erreurs permanentes. Une configuration peut avoir plusieurs distributions (plusieurs implémentations). Dans chaque distribution,

on insère une zone vide à une place différente. En fonction de la détection d'une faute, on change alors la distribution de la configuration courante de manière à ce que la zone vide recouvre la zone fautive. Ces approches nécessitent des ressources spécifiques pour la gestion de la reconfiguration.

Les approches traditionnelles de la détection et de la tolérance aux fautes aux niveaux composant et architectural s'appuient sur la notion de redondance. On distingue en général deux approches : la redondance double (DMR – Dual Modular Redundant) où l'on double la ressource de calcul, et la redondance triple (TMR). Le problème de la redondance vient de l'insertion d'un vote majoritaire afin de détecter un chemin défaillant. Dans les technologies ASIC ce circuit est durci (conçu pour résister aux SEU). Cependant, dans un FPGA cette approche est impossible. L'idée est alors de tripliquer les voteurs eux-mêmes. La méthode *TMR synchronisée* [71] combine la reconfiguration dynamique partielle avec la TMR. Un voteur spécial, capable d'indiquer quel est le module défectueux, a été développé. Grâce à la reconfiguration dynamique partielle, un bloc défectueux est reconfiguré et une re-synchronisation est effectuée pendant que le reste du système fonctionne. Pour les systèmes à base de processeurs, le système *lockstep* correspond à une implémentation DMR. Ce sont en fait deux cœurs de processeur qui exécutent le même ensemble d'opérations en parallèle et de manière strictement synchronisée. Les sorties des opérations ainsi effectuées peuvent être comparées afin de déterminer s'il y a eu une faute. Ce système a été implémenté par Xilinx autour de 2 PowerPCs [89].

Le problème majeur de ces approches est leur surcoût matériel important. Nous proposons donc une architecture tolérante aux fautes par application d'une redondance matérielle, mais dont les éléments servent aussi pour l'application.

D.1 Système multiprocesseurs tolérant aux fautes

D.1-1 Architecture

Notre système est construit en exploitant la reconfiguration dynamique des circuits FPGA de la série Xilinx Virtex. Un système dynamique typique est construit autour d'un microprocesseur qui lit les bitstreams de configuration dans une mémoire externe et contrôle l'interface de reconfiguration (ICAP) en envoyant les bitstreams partiels (PRM – Partially Reconfigurable Module dans la dénomination Xilinx) des différentes zones reconfigurables (PRR). Dans les approches classiques, ce processeur est intégré de manière statique pour contrôler les autres ressources de manière dynamique.

Dans notre système, ce processeur de contrôle est également un cluster reconfigurable dynamiquement. Ceci afin de garantir la possibilité de reconfigurer tous les processeurs en cas de défaillances. Notre système se compose de quatre cœurs MicroBlaze (figure 4-8). Les connexions entre processeurs sont garanties par des bus de type point à point (non représentés sur la figure pour des questions de clarté). Chaque processeur ainsi que ses périphériques est implémenté dans une PRR (nommé μC). Dans notre système, si la reconfiguration du processeur 2 est

requise, le processeur 1 contrôlera l'accès à l'interface ICAP et lui enverra le flux de bits approprié. Si c'est le processeur 1 qui doit être reconfiguré alors le processeur 2 prendra le contrôle de l'ICAP. C'est le même mécanisme entre les processeurs 3 et 4, qui peuvent se reconfigurer l'un l'autre. La reconfiguration d'un processeur peut avoir lieu lorsqu'un processeur devient fautif ou si nous voulons migrer certaines tâches à ce processeur.

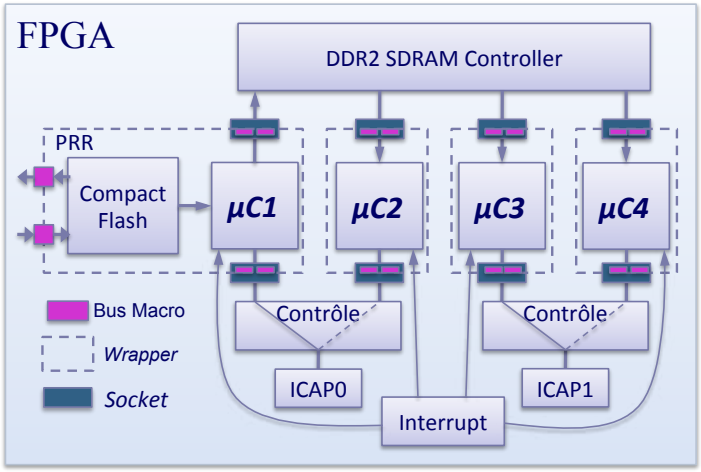


Figure 4-8 – **Architecture du système multiprocesseurs reconfigurable dynamiquement**. *Les différents processeurs peuvent se reconfigurer les uns, les autres. La gestion distribuée de la reconfiguration nécessite la mise en place d'une hiérarchie de la mémoire de configuration. Les interfaces des différents modules nécessitent la création de* wrappers *et de* sockets.

Les interfaces de reconfiguration sont quant à elles *mappées* dans la logique statique. Comme il n'y a que deux ICAP dans un Virtex 5 et que les 4 processeurs sont capables de les contrôler, deux contrôleurs supplémentaires sont nécessaires.

Le fait de rendre tous les processeurs reconfigurables nécessite de mettre en place un système de gestion de la reconfiguration distribué et non plus centralisé. Dans les systèmes classiques centralisés un seul MicroBlaze peut être connecté à la mémoire de configuration (CompactFlash figure 4-8) à cause des interfaces de sorties du circuit. Il faut alors mettre en place une hiérarchie mémoire de reconfiguration afin que les autres MicroBlazes puissent accéder à l'ensemble des bitstreams du système. Il est donc nécessaire de copier ces bitstreams dans une autre mémoire. En raison de la taille limitée des BRAM disponibles dans les FPGA, nous avons utilisé de la mémoire SDRAM DDR2 présente sur la carte que nous avons utilisée.

La reconfiguration dynamique permet de modifier le comportement du processeur à tout moment. Cependant, le besoin de connexion entre les processeurs nécessite la définition d'interfaces stables entre les modules reconfigurable (PRM). Afin de garantir cela, nous avons défini la notion de *wrapper* qui décrit toutes les interfaces d'un PRM avec la logique statique du circuit (notamment le bus d'interconnexion inter processeurs). Ce composant est un artefact permettant d'utiliser les outils de synthèse pour la reconfiguration dynamique. L'avantage de cette approche est que nous n'avons pas besoin de prendre en compte la complexité de la structure

d'un module reconfigurable, puisque chaque PRM qui sera implémenté dans un PRR aura les mêmes interfaces. Nous avons écrit quatre *wrappers* pour les quatre PRR de notre système afin que chaque processeur puisse être conçu de manière indépendante par rapport à son placement futur dans le système.

L'approche utilisée par les outils de conception nous ont amené à définir la notion de *socket*. L'objectif des *sockets* est de virtualiser des périphériques qui existent dans d'autres sous-systèmes. En effet, après le découpage du système, certains composants, tels que le contrôleur ICAP ou le contrôleur DDR2, ne sont plus connectés aux bus des MicroBlaze car ils appartiennent à la section statique du système. Ainsi, lors de la phase de conception des PRM, chaque processeur n'a plus de connexion directe avec ces contrôleurs. Cela génère des erreurs de synthèse matérielle car les connexions des différents bus ne doivent pas être ouvertes. Ce composant est aussi un artefact permettant de palier les manques des outils.

D.1-2 Gestion de la tolérance aux fautes

Régulièrement pendant les phases d'exécution, le système se synchronise via les bus inter processeurs. Pour éviter un délai de temporisation, nous utilisons une source d'interruption unique (figure 4-8) pour tous les processeurs. La période d'interruption est indépendante des horloges des processeurs. De cette manière, il est possible de synchroniser des processeurs fonctionnant à des fréquences différentes. La routine d'interruption permet alors de vérifier le bon fonctionnement de chaque processeur mais aussi des interconnexions. Les processeurs s'échangent des trames de vie pour vérifier le bon fonctionnement du système. En cas de désaccord entre les matrices de connexion ou de détection d'une défaillance dans un des processeurs, une phase de reconfiguration est lancée afin de voir si le processeur fautif redémarre. Pendant cette phase, deux processeurs continuent l'exécution de leurs tâches respectives pendant que les deux autres tentent de réparer le problème. Il reste ici des problèmes de hiérarchisation de tâches (afin de ne pas interrompre une tâche prioritaire) et de recouvrement de l'erreur. Nous proposons et étudions actuellement une solution basée sur des trames de vie comportant le contexte des tâches pour la correction d'une faute d'exécution d'une tâche. En cas d'erreur dans un processeur, deux cas sont alors possibles :

1) le processeur redémarre suite à sa reconfiguration ; l'autre processeur lui renvoi alors le dernier contexte sain sauvé à la synchronisation précédente (checkpointing et rollforward) [7] ;

2) le processeur fautif ne redémarre pas (cas d'une faute permanente) ; une décision de migration de la tâche peut être prise. Nous considérons que le code source de la tâche est accessible par tous les processeurs et le contexte de la tâche a été diffusé lors des précédentes trames de vie. Un processeur sain peut donc rapidement reprendre l'exécution de la tâche interrompue.

D.1-3 Implémentation et comparaison

Nous avons implémenté notre système multiprocesseurs tolérant aux fautes (FT-DyMPSoC) dans un circuit Xilinx Virtex XC5VSX50T. Comme on peut le voir (figure 4-9), ce circuit de taille moyenne supporte largement l'ensemble du système et permettrait l'implémentation de mécanismes avancés de tolérances aux fautes ou plus de processeurs. Le système complet utilise 67% des slices du FPGA, et 65% des mémoires BRAM.

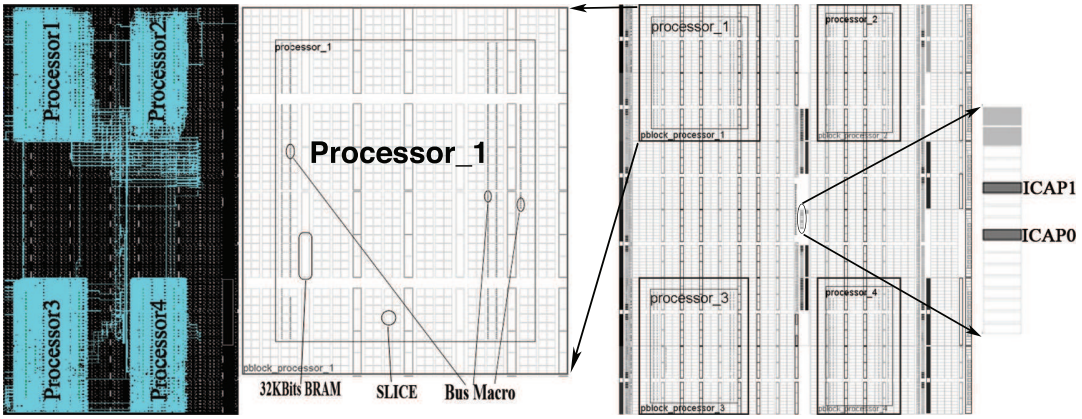


Figure 4-9 – **Floorplan du système implémenté sous PlanAhead**. *La logique de contrôle statique est très petite face aux processeurs. Chaque processeur est mappé dans une PRR et communique avec le reste du système au travers de bus macro. Les Virtex offrent à l'heure actuelle deux ICAP placées dans la colonne centrale du FPGA.*

On notera que les différences dans les tailles de bitstream, présentées dans la table 4-1, sont dues au placement des PRR dans le FPGA. Une attention particulière devra être apportée sur ce point car cela engendre des pénalités en terme de temps de configuration. Ceci est d'autant plus vrai si le PRR est à cheval sur différents domaines d'horloge du FPGA. Le processeur 1 est plus gros car il contient l'interface avec la mémoire CompactFlash.

	LUT	FF	Slice	BRAM	Taille du bitstream (Koctets)
Statique	7866	6616	2019	22	
Processeur 1	4160	4160	1040	16	253
Processeur 2	2880	2880	720	16	156
Processeur 3	3840	3840	960	16	198
Processeur 4	2880	2880	720	16	156

Tableau 4-1 – **Ressources consommées par le système**. *La partie statique inclut les contrôleurs d'ICAP, l'interface multi port pour l'accès à la DDR et le générateur d'interruptions.*

Comme nous l'avons dit plus haut, les bitstreams nécessitent d'être recopiés afin de les rendre accessibles à tous les processeurs. Nous avons ainsi évalué le temps de reconfiguration depuis la CompactFlash (CF2ICAP dans la Table 4-2). Nous avons alors mesuré le temps mis pour la recopie des bitstreams de la CompactFlash à la DDR2 (CF2DDR) et la durée de reconfiguration

du système depuis la DDR2 (DDR2ICAP).

Taille du bitstream (Koctets)	CF2ICAP (ms)	CF2DDR (ms)	DDR2ICAP (ms)
156	***326***	***281***	***80***
198	***414***	***357***	***101***
253	***529***	***456***	***129***

Tableau 4-2 – **Durée de la manipulation des bitstreams**. *Mesure des temps de transfert et de reconfiguration en fonction de la mémoire source et de la taille des bitstreams.*

On peut remarquer que la durée de reconfiguration depuis la DDR2 est 65% plus rapide qu'à partir de la CF. La première étape de recopie de bitstream est coûteuse (456+357 + 2 * 281 = 1,3 s), mais cette étape ne se déroule qu'une seule fois à l'initialisation du système. Les opérations de reconfiguration sont ensuite plus rapides. On gagnera alors du temps à chaque reconfiguration partielle du système. Nous avons pu valider, en utilisant d'autres tailles de bitstream, que le temps de reconfiguration augmente linéairement. Ceci est un argument de plus, motivant une conception attentive du système afin de réduire les bitstreams. Les approches utilisant de la redondance seront donc très pénalisantes.

Ces manipulations de bitstream peuvent augmenter les probabilités de défaillance et de changement de valeur d'un bit dans le flot de configuration. De plus, la mémoire DDR2 est plus sensible aux radiations que la mémoire CF. Pour répondre à ces problèmes, des codes détecteur et/ou correcteur d'erreurs peuvent être utilisés (calcul de parité, Hamming ou codes CRC). Toutefois, ces approches rajoutent des bits et peuvent donc augmenter la taille des bitstreams.

Dans notre système, l'intervalle minimal de synchronisation est de 130 ms (valeur supérieure au plus long temps de reconfiguration des processeurs=129 ms table 4-2). Le principe est de ne pas déclencher de synchronisation alors qu'un processeur est en cours de reconfiguration. En fonctionnement normal, les processeurs se reconfigurent depuis la DDR2 avec des durées réduites. La routine de synchronisation dans un système fonctionnant à 100 MHz a besoin de 287μs pour les quatre processeurs. Ce temps de synchronisation étant nettement inférieur à la durée entre chaque interruption, ce processus n'aura donc pas beaucoup d'impact sur les timings du système.

La table 4-3 compare différentes approches de gestion de la tolérance aux fautes dans les systèmes reconfigurables dynamiquement. La couverture de fautes évalue qualitativement le nombre de fautes potentiellement détectées par le système. La continuité de service représente la proportion de temps passé à la détection et à la correction de fautes, enfin la dernière colonne représente la capacité du système à supporter les erreurs permanentes.

Le scrubbing est sans conteste la solution la moins coûteuse, ce qui fait son succès. Cependant la continuité du service n'est pas assurée et les fautes permanentes ne sont pas supportées. De plus, cette technique a un impact non négligeable sur la performance du système puisque le FPGA est reconfiguré très périodiquement. Les techniques de redondance donnent de très bons résultats du point de vue de la couverture de fautes, mais leur coût d'implémentation est souvent

	Surcoût matériel	Impact performances	Couverture de fautes	Continuité du service	Fautes permanente
scrubbing	∼1x	+	100%	+-	non
TMR	>3x	- -	100%	+++ puis +-	non
DMR	>2x	- -	100%	-	non
FT-DyMPSoC	<1.6x	-	∼100%	+++	oui

Tableau 4-3 – **Comparaison de différentes technique de tolérance aux fautes**. *Le scrubbing est la technique la moins couteuse. La duplication offre une bonne couverture aux fautes. FT-DyMPSoC offre un compromis coût – couverture de fautes.*

prohibitif. FT-DyMPSoC est le seul système pouvant accommoder les fautes permanentes par utilisation de la technique de *tiling*. Son coût matériel est moyen pour une continuité maximale de service. Le taux de couverture est cependant inférieur aux autres techniques dû à la difficulté de détection des fautes dans les processeurs (une faute peut affecter un processeur sans pour autant l'empêcher de répondre à la synchronisation). Pour palier ce problème, nous proposons l'implémentation d'un système *lockstep* dynamique.

D.2 Système lockstep reconfigurable dynamiquement

Dans un système lockstep, les deux processeurs reçoivent les mêmes entrées et exécutent les mêmes instructions simultanément. Ce système souffre de deux inconvénients : (1) il ne permet pas de corriger une erreur et (2) le service est interrompu définitivement en cas d'erreur. L'utilisation de la reconfiguration dynamique permet de lever ces deux verrous.

Nous proposons une nouvelle architecture lockstep utilisant des processeurs "softcore". Afin d'identifier le cœur fautif, nous avons ajouté au système une machine de détection de fautes dans le plan de reconfiguration. Une fois le processeur fautif identifié, nous corrigeons l'erreur par reconfiguration dynamique de ce seul cœur, et nous appliquons une technique de recouvrement de type rollforward [74]. Dans le même temps, le service est maintenu par le processeur non fautif qui peut continuer son exécution.

D.2-1 Architecture

La figure 4-10 présente l'architecture complète du système. Il est constitué du lockstep amélioré (*Enhanced lockstep*), et de la gestion de la reconfiguration (*FT Configuration Engine*). La méthode s'appuie sur l'utilisation d'une mémoire de stockage des bitstreams sûre de fonctionnement. Cette propriété est assurée par l'utilisation de techniques classiques pour les mémoires (comme le codage qui sont en dehors de nos travaux).

Le "Enhanced lockstep" est construit autour de deux processeurs MicroBlaze μP1 and μP2 identiques et couplés entre eux (figure 4-10). En fonctionnement normal, leurs sorties sont

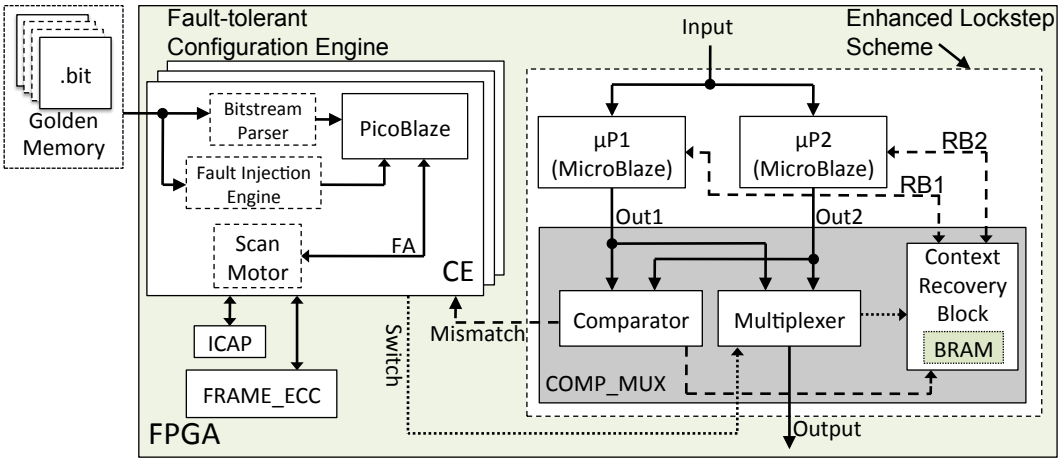


Figure 4-10 – **Architecture "Enhanced lockstep"**. *Une fois une faute détectée par le système lockstep, le "FT Configuration Engine" permet la localisation de celle-ci. Le processeur fautif est alors reconfiguré, puis resynchronisé avec l'autre processeur. Le service n'est pas interrompu, le processeur sain continuant l'exécution de l'application.*

identiques et entrent dans un comparateur. Une seule des deux sorties est utilisée par le reste du système. On notera que la synthèse de ces processeurs doit être soignée afin de bien séparer leurs ressources. Ceci afin de faciliter la localisation de la faute, et de ne pas perturber le fonctionnement du processeur sain pendant la reconfiguration de l'autre processeur en cas de faute.

La sortie du comparateur permet de détecter une erreur par différence entre les sorties (signal *Mismatch*). Hormis la comparaison, le bloc *COMP_ MUX*, intègre un multiplexeur qui connecte un des deux processeurs au reste du système en fonction de la localisation de la faute. Ce changement de sortie est atomique et est réalisé en un seul cycle d'horloge. Le composant *COMP_ MUX* est lui même intégré dans une PRR, et peut donc être reconfiguré en cas d'erreur. Cette opération suspend l'exécution des processeurs, et donc l'application.

Une fois l'erreur localisée par le *FT Configuration Engine* (présenté ci-dessous), un processus de reconfiguration est lancé afin d'éliminer l'erreur. Une fois la reconfiguration effectuée, les deux cœurs sont (re-)synchronisés afin de mettre le processeur reconfiguré dans le même état que le processeur sain. Cette opération est effectuée par le bloc *Context Recovery*, qui permet le stockage local du contexte du processeur sain et gère ensuite l'envoi de ce contexte à l'autre processeur (bus RB1 et RB2).

Lors de la détection d'une différence entre les deux processeurs, la gestion de la configuration lance la recherche de localisation de l'erreur par relecture du plan de configuration. Le *Fault-Tolerant Configuration Engine* est le cœur du système de détection. Il nécessite donc d'être sûr de fonctionnement. Ce bloc étant très petit en comparaison des MicroBlaze, nous avons utilisé la technique de triplication synchronisée [71] (le *Fault-Tolerant Configuration Engine* est donc composé de trois *Configuration Engine* (figure 4-10) et d'un voteur non représenté). En cas de

détection d'erreur dans ce bloc, la branche fautive est reconfigurée afin de conserver un haut niveau de fiabilité.

Chaque *Configuration Engine* est constitué d'un moteur de recherche *Scan Motor*, d'un parser de bitstream *Bitstream Parser* et d'un processeur PicoBlaze (choisi pour sa petite taille). Le *Scan Motor* est un process continu qui scanne en permanence la mémoire de configuration et vérifie toutes les "frames" du circuit. C'est lui qui permet de localiser l'erreur, même celles n'affectant pas directement les "Enhanced lockstep". Ce composant utilise l'ICAP pour faire du "readback" du bitstream et le *FRAME_ECC* [91] qui est une primitive Xilinx permettant la détection et la correction d'erreurs dans les "frames" de configuration. Une fois l'erreur trouvée, il passe l'adresse de cette dernière au PicoBlaze afin de déterminer le composant fautif. Ne possédant pas d'outils permettant d'extraire un composant à partir d'un bitstream, nous avons implémenté le *Bitstream Parser* qui sauvegarde une table des adresses des frames utilisées par chaque composant. Cette table est générée, au démarrage du système, par lecture des différents bitstreams partiels. Ce bloc est actuellement implémenté sous forme logicielle dans le PicoBlaze.

D.2-2 Gestion de la tolérance aux fautes

Lorsque *COMP_MUX* détecte une erreur, il demande un scan du "Enhanced lockstep". Le "Fault-Tolerant Configuration Engine" commence par scanner le comparateur, et le reconfigure immédiatement en cas d'erreur. La localisation de la faute ne nécessite alors le scan que d'un seul processeur. Une fois le composant possédant la "frame" défaillante localisé, la sortie du "Enhanced lockstep" est aiguillée sur le processeur sain. La suite de la procédure est alors transparente du point de vue de l'application. Le PicoBlaze essaye alors de corriger uniquement la "frame" fautive (en utilisant le *FRAME_ECC* basé sur le code SEC/DED [1] de Hamming). Si l'erreur persiste, tout le composant est alors reconfiguré entraînant une latence de reconfiguration plus grande. Si l'erreur persiste après cette deuxième phase de correction, alors l'erreur est déclarée permanente. Nous utilisons dans ce cas là, la technique de *tiling* en configurant une distribution différente du processeur fautif.

Après une phase de reconfiguration, le processeur se retrouve dans son état de démarrage. Il faut alors le remettre dans l'état courant du système. Nous utilisons une stratégie de rollforward pour ce faire. Le contexte d'un MicroBlaze est constitué de 32 registres banalisés et de deux registres spéciaux (chacun sur 32 bits). Dès la fin de la reconfiguration du processeur fautif, un signal indique que ce dernier est prêt pour une re-synchronisation. L'instant de cette synchronisation est décidé par le processeur non fautif afin de ne pas interrompre une tâche critique. Ce processeur indique alors au *Context Recovery* quand démarrer l'échange de contexte, ce qu'il fait en déclenchant une interruption. Pendant cette phase, le service est suspendu.

1. Single Error Correction / Double Error Detection

D.2-3 Implémentation et comparaison

Nous avons implémenté le système complet dans un Xilinx Virtex-5 XC5VSX50T (figure 4-11). La table 4-12 présente l'utilisation des ressources du FPGA pour chaque bloc constitutif du système (les résultats sont donnés pour un processeur et un seul *Configuration Engine*). Chacun des trois *Configuration Engine* est reconfigurable dynamiquement. L'ensemble du système supporte les erreurs temporaires, sauf le voteur. Ce composant n'occupant que 20 slices, la probabilité d'une erreur est cependant faible.

Les deux MicroBlazes fonctionne à 125MHz, tandis que les trois PicoBlazes ont une fréquence de 50MHz. Le *COMP_MUX*, intègre 4Ko de mémoire double port pour l'échange de contexte.

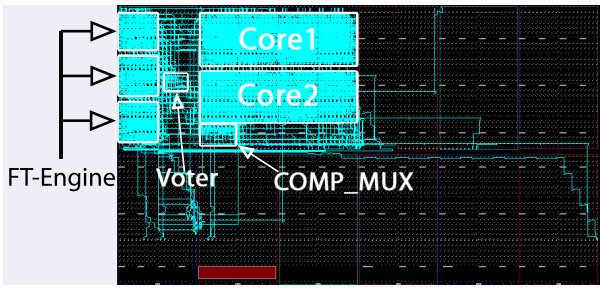


Figure 4-11 – ***Floorplan du système.***

Module	Slices	BRAM [Ko]
Configuration Engine	120	
Voteur	20	
MicroBlaze	560	32
COMP_MUX	60	4
XC5VSX50T	32640	132

Figure 4-12 – ***Utilisation de ressources pour chaque bloc du système.***

Le surcoût matériel du *FT Configuration Engine* est donc de $3 \times 120 + 20 = 380$ slices. Ce coût est constant et n'apparaît qu'une seule fois dans le système. Il est alors possible d'implémenter un système multiprocesseurs (comme FT-DyMPSoC) durci pour un coût supplémentaire de 60 slices pour le *COMP_MUX*. Ce surcoût est calculé par rapport à une implémentation de processeur utilisant la technique de lockstep « classique ».

Le scan d'une frame nécessite 41 cycles d'horloge. La détection d'une erreur nécessite de scanner le *COMP_MUX* et un processeur, constitué respectivement de 30 et 330 frames de configuration. Le temps de détection pour le *COMP_MUX* est alors de $t_{sCM} = 30 \times 41 \times 20$ ns $= 25$ μs, et le temps de scan d'un processeur est de $t_{sP} = 270$ μs. Le temps de reconfiguration de *COMP_MUX* est de $t_{rCM} = 185$ μs. Le processus de re-synchronisation entre les processeurs nécessite quand à lui $t_{rec} = 5.2$ μs. De ce fait, le temps maximum pendant lequel l'application est suspendue à cause d'une erreur est de $t_{i_1} = 25 + 185 = 210$ μs — si l'erreur apparaît dans le bloc *COMP_MUX*, et $t_{i_2} = 25 + 270 + 5.2 = 300$ μs — si c'est un processeur qui est fautif. Le temps de reconfiguration d'un processeur (en ms cf. table 4-2) ne rentre pas en ligne de compte puisque ce temps est masqué par l'exécution de l'application par le processeur sain.

Afin de comparer nos résultats, nous avons implémenté aussi un lockstep classique pour les MicroBlazes (i.e. sans la détection et la correction de fautes). Ce système "classique" utilise les mêmes ressources que notre système car il faut intégrer en plus un processeur de contrôle de la reconfiguration. Nous avons choisi un PicoBlaze pour cela, mais devant être lui même

tolérant aux fautes nous avons utilisé la technique de TMR. Du point de vue temporel, notre système est largement meilleur en terme de temps d'interruption de service. En effet, dans le système classique, dès qu'une faute est détectée il faut reconfigurer le système lockstep (les deux processeurs, ce qui prend plusieurs ms). De plus, la stratégie de restauration de contexte par rollforward est beaucoup plus efficace que la stratégie de "checkpointing and rollback" nécessaire dans le système classique. Le processus de re-synchronisation est donc lui aussi beaucoup plus efficace.

Une approche par utilisation de la TMR [40] permet de rendre tolérant aux fautes un système à base de MicroBlaze. Le surcoût matériel de notre système est largement inférieur à cette approche (1×Microblazes +*FT Configuration Engine*= 940 slices vs 2×Microblazes+Voteurs=1180 slices). Ceci est d'autant plus critique si l'on souhaite réaliser un système multiprocesseurs. Cependant, l'approche TMR offre l'avantage de n'avoir aucune interruption de service. Cependant, la technique présentée dans [40] nécessite un ordinateur externe pour contrôler la reconfiguration, alors qu'elle est supportée en ligne dans notre système.

E Conclusions et perspectives

Il est notable que ce domaine de la gestion de la reconfiguration dynamique est très actif. Loin de vouloir être exhaustif il n'y a actuellement, à ma connaissance, aucun projet s'intéressant à la problématique de manière globale, c'est à dire prenant en compte les aspects architectures et logiciel d'un SoC reconfigurable.

Une piste intéressante de développement pour la validation et l'exploration d'une plate-forme reconfigurable dynamiquement est l'étude des approches ligne de produit (DSPL – Dynamic Software Product Lines) [69]. Nous avons entamé des collaborations avec l'équipe Triskell de l'IRISA dans ce sens.

Nous avons défini les avantages apportés par un OS embarqué sur un RSoC, ainsi que les contraintes imposées par celui-ci à la fois sur le déploiement d'une application, et sur la conception de la plate-forme elle-même. Les premiers travaux menés dans ce sens ont permis d'établir les liens et les interactions entre l'OS (et ses services) et les différents éléments reconfigurables de la plate-forme. Nous avons ainsi formalisé l'ensemble de la modélisation d'une telle plate-forme. Au travers de cette modélisation, nous avons alors commencé à proposer des services dédiés et spécifiques même si leurs définitions n'est qu'embryonnaire. Nous développons actuellement un ensemble de services (ordonnancement spatio-temporel, gestion des communications) dont les résultats préliminaires sont très encourageants. De plus, la modélisation et les développements faits pour l'outil Dogme nous permettront d'avoir des bases de comparaisons pour nos développements futurs.

Supporter la tolérance aux fautes est une nécessité pour l'avenir et pour le développement de l'utilisation des ARD dans de nouveaux domaines. Nous avons ainsi proposé un système

multiprocesseurs tolérant aux fautes et nous avons étendu le concept de *lockstep* pour des processeurs "soft-core". Nous travaillons sur des mécanismes d'injection de fautes afin de montrer l'efficacité de nos différentes techniques et de les comparer avec l'état de l'art. Nous injectons des fautes par la modification d'un bit d'une frame de reconfiguration en utilisant une approche de *scrubbing*. Cette technique permet d'être embarquée dans le système afin de diminuer les temps de validation et de test. Les premiers résultats sur notre système "Enhanced lockstep" montre l'efficacité de ce dernier.

Nous passerons ensuite à l'étude de la tolérance aux fautes au niveau système. A ce niveau, nous avons lancé l'étude d'une plate-forme comportant plusieurs FPGA dans laquelle chaque circuit contient un FT-DyMPSoC. En garantissant le bon fonctionnement de chaque FPGA, par les mécanismes de sûreté de fonctionnement étudiés précédemment, on peut assurer la tolérance aux fautes de l'ensemble de la plate-forme. Nous considérerons alors le cas où un FPGA tombe en panne et nous proposerons des mécanismes permettant à la plate-forme de maintenir les services vitaux de l'application. Pour cela, les autres FPGA devront maintenir les fonctionnalités du FPGA défaillant par redéploiement des tâches de l'application. Les mécanismes de détection et de correction de fautes seront adaptés et implémentés au niveau système.

Chapitre

5 Bilan et perspectives de recherches

Résumé : J'ai présenté tout au long de ce document les perspectives à court et moyen terme de mes différents travaux. Je dresse ici, sur la base d'un bilan, les perspectives à plus long terme que je souhaite aborder dans les années à venir.

Sommaire

A Bilan scientifique

Dans ce document, j'ai présenté onze années de mes activités de recherche. Je présente ici la mise en perspective de l'ensemble de ces travaux. Je tiens à préciser que dans mon analyse la chronologie des différents projets n'est pas forcément respectée.

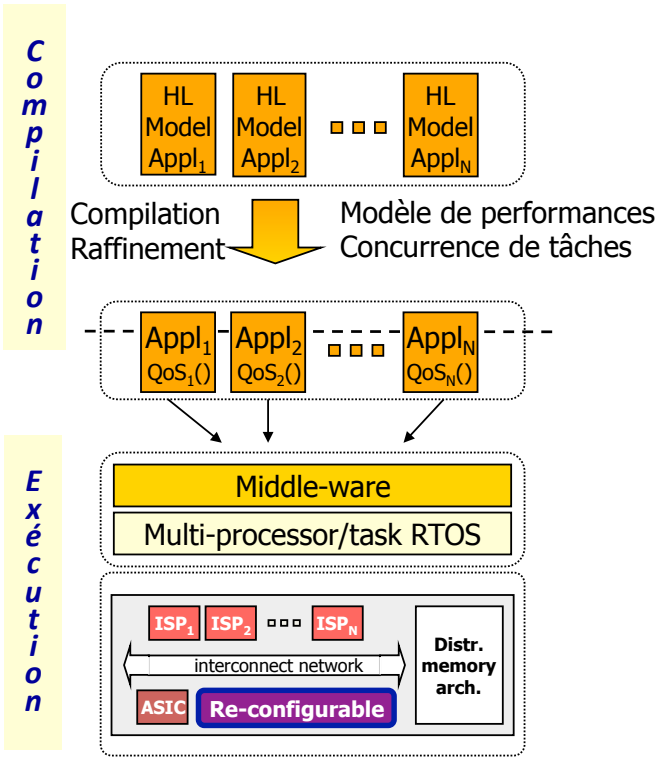


Figure 5-1 – **Vision de mon activité de recherche**. *Clairement positionnée dans le développement de RSoC, j'ai développé des activités sur l'ensemble du flot de conception d'une ARD. J'ai aussi bien travaillé sur les aspects architecture que sur les aspects conception/compilation. Enfin, j'ai proposé le développement de méthodologies permettant de simplifier l'utilisation des ARD, ou encore de prendre en compte des aspects tolérance aux fautes.*

Mon activité de recherche a débuté en 1995 lors de mon D.E.A. au Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier (LIRMM) à peu près à la même époque où naissait le concept de reconfiguration dynamique avec le Xilinx XC6200. Dès lors, ce concept a guidé la majorité de mes travaux. En son époque, la reconfiguration dynamique souffrait de deux inconvénients majeurs. Un manque d'outils et de méthodes permettant une utilisation simple et efficace de la puissance potentielle des architectures, et un manque de besoins applicatifs. Ces deux inconvénients se sont résorbés au fil des ans, avec notamment l'apparition de nouvelles applications comme la réalité augmentée, ainsi que l'apparition d'outils qui, bien qu'imparfaits, ont le mérite d'exister. J'ai, tout au long de ces onze années, essayé de contribuer à palier ces inconvénients.

Mes activités de recherche lors de la préparation de mon diplôme de doctorat (1995-1998) s'inscrivaient dans le cadre de la conception de systèmes numériques intégrés sur puce. La complexité de ces systèmes a conduit à proposer de nouvelles méthodologies de conception. En particulier, mon travail sur le prototypage est devenue une étape clé de la conception et de la validation, et m'a amené à concevoir un nouvel environnement de prototypage construit autour d'une plate-forme reconfigurable. Ce concept s'appuyait sur des composants d'interconnexions programmables permettant de connecter, dans des zones banalisées, des modules de natures différentes (DSP, mémoires, FPGA, . . .). Mes travaux visaient à proposer des estimations et des prédictions de performances des systèmes à partir d'une intégration sur prototype programmable. Fort de cette expérience, j'ai alors commencé à proposer des mécanismes architecturaux permettant de supporter efficacement le principe de la reconfiguration dynamique.

Maître de conférences depuis septembre 1999 à l'IUT de Lannion (Université de Rennes 1), j'ai intégré l'équipe Signal et Architecture du LASTI, devenue l'équipe projet INRIA CAIRN. A mon arrivée dans l'équipe, je me suis plus particulièrement intéressé aux nouvelles méthodologies de conception. J'ai alors travaillé sur différentes approches visant la conception orientée plate-forme par l'utilisation de blocs IP de haut niveau, ou par la définition de flots de conception "génériques". Ces travaux m'ont convaincu de l'importance d'offrir des mécanismes de gestion des architectures afin de permettre la réutilisation de flot de conception en s'abstrayant de la complexité matérielle sous-jacente.

Une première analyse de mes travaux montre qu'à la faveur de nouvelles évolutions technologiques, j'ai été régulièrement amené à développer de nouveaux concepts architecturaux. De plus en plus complexe, ou offrant de nouveaux services, chacun de ces concepts vise à tirer partie de la dynamicité du monde environnant. Une deuxième conclusion est que nous sommes aujourd'hui à un croisement. Face aux évolutions technologiques, et à la fin programmée des technologies CMOS, et les nouveaux besoins applicatifs (dynamicité, flexibilité, comsommation, ...), les architectures futures seront en rupture avec les systèmes actuels. Ces architectures et leurs flots de conception restent à imaginer, mais devront, quoi qu'il arrive, supporter la reconfiguration, voire l'adaptabilité. Enfin, développer une architecture n'est pas suffisant si les méthodes de conception et de gestion associées ne sont pas efficaces. Ce dernier point nécessite donc la création de méthodologies conjointement à la mise en place de paradigmes architecturaux.

B Perspectives de recherches

Les systèmes multi-cœurs offrent un nouveau degré de liberté pour s'adapter dynamiquement à leur environnement. Cette adaptation est possible par la reconfiguration de chaque processeur (qu'il soit à usage général, ou un bloc spécifique). La configuration manuelle de tels systèmes est alors difficile, voire impossible si le nombre de processeurs augmente. Ces systèmes devront donc s'adapter d'eux mêmes, de manière autonome. La multiplicité des objets, virtuels et

matériels, dont seront constitués ces systèmes nécessitera de repenser certaines dimensions de l'informatique. Ainsi l'interopérabilité est certainement une des notions qui est amenée à se développer de manière importante dans les années à venir. La synthèse de mes travaux, ainsi que mon analyse actuelle, m'amènent naturellement à imaginer des systèmes autonomes, homogènes et interopérables. Cette synthèse mène à la définition de systèmes auto-adaptatifs. Cette évolution est représentée sur la figure 5-2, et montre alors l'articulation de mes travaux passés, présents et futurs.

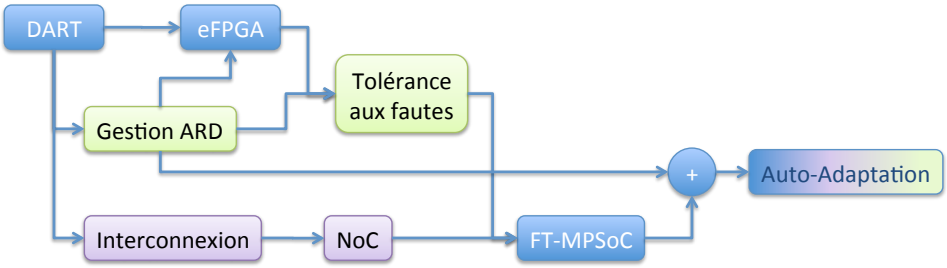


Figure 5-2 – **Vision de mon activité de recherche future**. *Des architectures reconfigurables dynamiquement vers les systèmes adaptatifs.*

Les techniques permettant l'adaptation comme l'auto-diagnostic, l'auto-configuration, et l'auto-optimisation sont étudiées depuis plusieurs années dans le domaine du calcul bio-inspiré ou du calcul organique [78]. Cependant, si ces méthodes commencent à être efficaces, leur implémentation dans un système embarqué et temps réel n'est pas encore assurée et un grand nombre de verrous subsistent. Ainsi, la conception d'applications pour des systèmes multi-cœurs embarqués doit prendre en compte les défis de l'adaptabilité, de la reconfiguration, du support du parallélisme (notamment distribué) et de la gestion de l'hétérogénéité. L'aspect enfoui de ces futurs systèmes nécessite, de plus, de s'attaquer au mur de la consommation, qui est de fait la contrainte majeure de mes travaux.

J'axe donc les perspectives de mon travail afin d'adresser cette problématique, selon cinq axes ayant des échéances à plus ou moins long terme.

Architectures reconfigurables dynamiquement pour la tolérance aux fautes

L'utilisation des architectures reconfigurables dans les applications critiques comme les transports ou l'espace devient une nécessité. Malheureusement comme nous l'avons vu, les architectures actuelles sont sujettes aux fautes, notamment celles générées par les radiations ou le bruit électromagnétique ambiant. La variabilité de la technologie est un autre facteur qui va influencer ces systèmes. En effet des erreurs permanentes dues au vieillissement, ou des fautes transitoires vont apparaître tout au long de la vie du circuit.

Je vais donc continuer mes travaux autour de cette problématique en étudiant à la fois des méthodologies pour architectures grain-fin, mais aussi en étudiant l'apport des architectures

gros-grain (telle que DART) dans ce domaine. La tolérance aux fautes doit être assurée à un coût raisonnable, en utilisant des techniques de détection simples (duplication, codes détecteurs d'erreurs ou conception d'opérateurs ad'hoc) et des techniques de recouvrement efficaces (rollforward, scrubbing, migration de tâches). Si l'aspect recouvrement est relativement bien étudié à l'heure actuelle, l'aspect détection de fautes reste à développer. Nous souhaitons donc étudier des techniques arithmétiques permettant une détection rapide des erreurs dans le cadre d'une architecture gros-grain. L'utilisation de bases de représentation numérique différentes (système redondant par exemple), l'algorithmie arithmétique et la conception adaptée des opérateurs peuvent permettre une détection, voire une correction, des fautes à un coût inférieur par rapport aux techniques actuelles. L'utilisation de la reconfiguration (comme présentée dans ce document) combinée à ces techniques de détection permettront alors d'envisager une architecture tolérante aux fautes autonome (*self-healing* i.e. corrigeant elle même son fonctionnement en présence d'erreurs).

Architecture multiprocesseurs adaptable

L'évolution actuelle des technologies d'intégration nous amène vers des approches de conception basées sur le parallélisme. Le moteur actuel de l'intégration et de la performance se base sur l'intégration de nombreux processeurs (dédiés ou généralistes) sur une même puce (approche MPSoC). Les modèles actuels sont tous basés sur l'utilisation de cœurs de calcul homogènes afin de "simplifier" la mise en œuvre de ces architectures (la réalisation d'un circuit homogène est plus simple) et de permettre une programmation plus aisée. Malheureusement, cette approche homogène ne permet souvent pas d'atteindre les performances requises, et notamment en terme de consommation. De plus, ces systèmes embarqués devront interagir avec leur environnement afin de s'adapter au monde extérieur ou devront supporter des évolutions à la demande des utilisateurs. La flexibilité sera alors le moteur de développement économique de ces MPSoC.

La solution que je souhaite étudier, est alors d'intégrer au sein de calculateurs élémentaires, des accélérateurs reconfigurables permettant le saut en performances et en flexibilité requis. Ce type d'approche permettra de supporter de l'hétérogénéité temporelle, alors que l'architecture "spatiale" est homogène. Les verrous majeurs sont alors : (1) l'étude du couplage entre les cœurs de calcul et leurs accélérateurs reconfigurables, (2) l'infrastructure de communication (devant véhiculer les données mais aussi le contrôle (reconfiguration)) et (3) les flots de conception. Sur ce dernier point, la vérification/validation du système qui doit avoir lieu le plus tôt possible dans le flot de conception, est certainement le point dur. Ce type d'architecture homogène supportera la migration de tâches entre éléments de calcul et permettra l'adaptation autonome des performances globales de l'architecture.

Architecture virtuelle

Les architectures reconfigurables constituent un support privilégié pour l'accélération matérielle de traitements car elles offrent une grande flexibilité. Le circuit correspondant à la partie de l'application accélérée peut être dynamiquement configuré complètement ou partiellement à la demande, et à distance. Cependant, l'inconvénient majeur est de devoir prédéterminer à la compilation les zones reconfigurables partiellement, et de devoir utiliser des outils logiciels complexes pour la programmation de ces zones. Ces contraintes pénalisent l'usage des ressources reconfigurables dynamiquement à l'image de la gestion de la mémoire (relocalisation, défragmentation, swap, segmentation).

Nous proposons de nous affranchir de ces contraintes par la définition d'une « architecture reconfigurable virtuelle ». L'objectif est de définir l'architecture et les opérations de bases devant y être intégrées pour permettre une gestion système : homogénéité (favorisant la relocalisation de tâches) et mécanismes servant de support aux primitives du système d'exploitation (commutation rapide de contexte, verrou, granularité de la page de configuration, etc.). Cette architecture virtuelle sera alors portée sur des architectures du commerce de type FPGA, ou sur une architecture spécialement conçue enchâssant les primitives de gestion. Les mécanismes avancés de gestion vont permettre un déploiement dynamique d'applications sur l'architecture. Nous envisageons donc d'inclure dans la machine virtuelle un sous ensemble des outils de placement-routage permettant de réaliser une synthèse dynamique pour la gestion du placement des tâches (Just In Time compilation). La machine virtuelle devra aussi être capable d'estimer son environnement et de prendre des décisions de mapping d'opérateurs. Des métriques et des politiques de gestion évoluées doivent donc être imaginées. Le monitoring de l'application, de l'architecture et de l'environnement est à étudier. Enfin, il faudra imaginer les mécanismes permettant de représenter ces contraintes pour les nouvelles applications non définies lors de la conception de l'architecture.

Architecture pour les futures technologies

Les innovations technologiques apportent continuellement de nouvelles opportunités de conception pour les systèmes embarqués. La première innovation importante, déjà disponible, est la superposition de couches pour la définition d'architectures 3D (3D stacking). Cette approche superpose différentes technologies dans un même circuit réalisant un SIP (System in Package). Cette technologie pourrait alors être le support de l'architecture multiprocesseurs adaptable présentée ci-dessus, en intégrant les éléments (processeurs, blocs reconfigurables, mémoires, réseau) dans des niveaux différents. Ceci permettrait une homogénéité dans chaque couche, et le partage de ressources si l'architecture intègre un NoC flexible. La deuxième problématique de ces architectures est justement le routage. L'aspect 3D permet de nouvelles stratégies, notamment la migration de tâches, pour palier les problèmes de réchauffement de la puce. L'interconnexion va, là encore, devoir supporter de nouvelles contraintes, et le crosstalk inductif devra être pris

en compte. Enfin, la gestion dynamique de ces architectures va devoir intégrer cette nouvelle dimension, notamment pour les aspects placement de tâches.

À plus long terme, il s'agira de tirer profit des spécificités relatives aux innovations technologiques pour définir de nouveaux paradigmes de calcul et de nouvelles méthodologies de conception. L'émergence des mémoires magnétiques, des solutions basées sur les nano tubes de carbone voire de l'électronique moléculaire laisse en effet présager que la régularité des structures de calcul pourrait être la clé de la valorisation de ces technologies. Cependant, la variabilité dans ces *process* rendra obligatoire les approches de tolérance aux fautes qu'il faudra inventer pour ces technologies. La technologie CMOS affiche dès aujourd'hui ses limites et la communauté scientifique internationale travaille ardemment à la conception de nouvelles technologies. En conséquence, de nombreuses solutions sont envisagées (nanotubes de carbones, dispositifs à 1 électron, points quantiques, ...) et il est impossible, dans l'état actuel de nos connaissance de préjuger des solutions d'avenir. En tout état de cause, il convient de se préparer à ce qui sera probablement une révolution technologique. Parallèlement aux actions de recherches évoquées précédemment, il conviendra donc d'assurer une veille technologique afin de se préparer à relever le défi qui consistera à définir les nouveaux paradigmes de calcul (approche par calcul Bayésien par exemple) qui permettront l'exploitation efficace de ces nouvelles technologies.

Méthodologie de conception basée sur les modèles

La conception d'un SoC nécessite d'implémenter efficacement des applications dynamiques sur des architectures flexibles et parallèles, ce qui induit une complexité qui n'est pas supportée actuellement. Il est donc nécessaire de repenser les méthodologies et d'utiliser des approches basées sur les modèles afin de répondre à l'enjeu de la conception des futurs systèmes embarqués. Le défi est alors de développer des modèles possédant la sémantique nécessaire à la représentation du problème. De fait, le portage d'une application dynamique sur un SoC auto-adaptable, met en jeu de nombreux modèles associés à chaque étape du flot de conception. D'une part des modèles au niveau applicatif (modèle de calcul ou *Model of Computation*), utilisés pour spécifier les traitements constitutifs de l'application ainsi que leurs interactions (*Kahn Process Network*, *Synchronous Dataflow*, etc.). D'autre part, des modèles de plate-forme d'exécution, utilisés pour spécifier les composants du système à concevoir (ressources de traitement, de stockage, de communication). Enfin, des modèles permettant d'appréhender l'aspect dynamique des applications et des architectures reconfigurables. Si des travaux ont démarré sur ce thème, il n'existe pas, à l'heure actuelle, de solution prenant en compte l'aspect reconfigurabilité à la fois logicielle et matérielle de l'architecture. Si les approches de Génie Logiciel dites MDE (Model Driven Engineering) connaissent un succès certain dans le monde "logiciel", leur utilisation dans un contexte d'outils de conception de SoC n'est que relativement récente. Il reste de fait de nombreux problèmes ouverts, et en particulier ceux liés à la conception d'architectures auto-adaptables.

J'envisage donc la conception d'un atelier logiciel de conception conjointe logiciel-matériel basé

sur l'ingénierie des modèles. L'objectif est de proposer un environnement unifié basé sur différents modèles permettant d'appréhender les propriétés de chaque composant constitutif du domaine. L'utilisation de modèles spécifiques pour la partie applicative et pour la partie architecturale permettra de représenter de manière efficace les objets. La problématique majeure concernera donc la création d'équivalence entre modèles et d'assurer un ensemble d'outils permettant une évaluation et une exploration du portage d'une application vers une instance d'architecture.

Si cette liste découle de mon expérience, il est évident que je ne souhaite pas limiter mes activités futures à un ensemble de thèmes de recherches personnel. Cette liste se devra bien évidemment d'être confrontée à la réalité du temps, des moyens et des potentielles collaborations qui seront nécessaires. Cette analyse de mes activités futures et passées n'est certainement pas le terme de mes réflexions, et cet ensemble de perspectives sera amené à évoluer avec la suite de ma carrière. Je souhaite donc terminer ce manuscrit en citant François Jacob : "*L'imprévisible est dans la nature même de l'entreprise scientifique. Si ce qu'on va trouver est vraiment nouveau, alors c'est par définition quelque chose d'inconnu à l'avance*".

Bibliographie personnelle

Thèses et habilitations à diriger des recherches

[Pil98] S. Pillement, *Méthodologies d'évaluation et de prototypage des systèmes numériques intégrés*, thèse de doctorat, Université de Montpellier II, décembre 1998.

Articles de revues internationales

[CPS10a] D. Chillet, S. Pillement, O. Sentieys, « Real-Time Scheduling on Heterogeneous SoC Architectures Using Inhibitor Neurons in a Neural Network », *Journal of Systems Architecture accepted, in progress*, 2010.

[DSP+10] L. Devaux, S. B. Sassi, S. Pillement, D. Chillet, D. Demigny, « Flexible interconnection network for dynamically and partially reconfigurable architectures », *International Journal on Reconfigurable Computing 2010*, 2010, p. 15 pages.

[PPS10a] S. Piestrak, S. Pillement, O. Sentieys, « Comments on "A Low-Power Dependable Berger Code for Fully Asymmetric Communication" », *IEEE Communications Letters 14*, 8, août 2010, p. 1–3.

[PPS10b] S. Piestrak, S. Pillement, O. Sentieys, « On designing Efficient Codecs for Bus-Invert Berger Code for Fully Asymmetric Communication », *IEEE Transactions on Circuits and Systems, part II accepted, In process*, 2010.

[LPS09a] J. Lallet, S. Pillement, O. Sentieys, « Efficient and Flexible Dynamic Reconfiguration for Multi-Context Architectures », *Journal of Integrated Circuits and Systems 4*, 1, 2009, p. 36–44.

[MHV+09] B. Miramond, E. Huck, F. Verdier, A. Benkhelifa, B. Granado, T. Lefebvre, M. Aïchouch, J. Prevotet, Y. Oliva, D. Chillet, S. Pillement, « OveRSoC : a

Framework for the Exploration of RTOS for RSoC Platforms », *International Journal of Reconfigurable Computing 2009*, 2009, p. 22 pages.

[PPS09] S. Pillement, J. Philippe, O. Sentieys, « Spatio-temporal Coding to Improve Speed and Noise Tolerance of On-chip Interconnect », *Microelectronics Journal 41*, 2009, p. 480–486.

[PSD08] S. Pillement, O. Sentieys, R. David, « DART : A Functional-Level Reconfigurable Architecture for High Energy Efficiency », *EURASIP Journal on Embedded Systems (JES) 1*, 2008, p. 1–13.

Articles de revues nationales

[DPCD10a] L. Devaux, S. Pillement, D. Chillet, D. Demigny, « DRAFT : réseau flexible pour architecture reconfigurable dynamiquement », *accepté à Technique et Science Informatiques*, Hermes Science Publications, 2010.

[CPS09] D. Chillet, S. Pillement, O. Sentieys, « Ordonnancement de tâches par réseaux de neurones pour architectures de SoC hétérogènes », *Traitement du Signal, 26*, Lavoisier, 2009, p. 77–89.

[PD07] S. Pillement, R. David, « Architectures reconfigurable faible consommation – réalité ou prospective ? », *Technique et Science Informatiques, numéro spécial SoC, 26*, Hermes Science Publications, 2007, p. 595–622.

[DLP05] R. David, D. Lavenier, S. Pillement, « Du microprocesseur au circuit FPGA : une analyse sous l'angle de la reconfiguration », *Technique et Science Informatiques, 24*, Hermes Science Publications, 2005, p. 395–422.

Chapitres de livre et recueil d'articles sélectionnés

[CPS10b] D. Chillet, S. Pillement, O. Sentieys, *Algorithm-Architecture Matching for Signal and Image Processing*, Springer Verlag, 2010, ch. RANN : A Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable System-on-Chip, p. To appear.

[DPS04] R. David, S. Pillement, O. Sentieys, *Low Power Electronics Design, Computer Engineering, Vol 1*, CRC Press, août 2004, ch. Energy-Efficient Reconfigurable Processors, p. 20.1–20.15.

[DCPS02a] R. David, D. Chillet, S. Pillement, O. Sentieys, *SOC Design Methodologies, International Federation for Information Processing, 218*, Kluwer Academic Publishers, 2002, ch. A Dynamically Reconfigurable Architecture for Low-Power Multimedia terminals, p. 51–62.

[RSC+99a] S. Raimbault, G. Sassatelli, G. Cambon, M. Robert, S. Pillement, L. Torres, *VLSI : Systems on a Chip, 162*, Kluwer Academic Publishers, 1999, ch. Embedded sys-

tems design and verification : Reuse oriented prototyping methodologies, p. 407–414.

[PTRC98] S. Pillement, L. Torres, M. Robert, G. Cambon, *CODESIGN conception conjointe logiciel-matériel*, *Collection Technique et Scientifique des télécommunications*, Eyrolles, 1998, ch. Aide à la validation de systèmes logiciels/matériels, p. 189–200.

[PTRC97a] S. Pillement, L. Torres, M. Robert, G. Cambon, *Models in System Design*, *Current Issues In Electronic Modeling*, *9*, Kluwer Academic Publishers, avril 1997, ch. LIRMM : prototyping platform for hardware / software codesign, p. 103–113.

Conférences internationales avec actes

[JPSP10] S. Jafri, S. Piestrak, O. Sentieys, S. Pillement, « Design of a Fault-Tolerant Coarse-Grained Reconfigurable Architecture : A Case Study », *in : IEEE International Symposium on Quality Electronic Design (ISQED)*, San Jose, USA, march 2010.

[PDP10] M. Pham, L. Devaux, S. Pillement, « Dynamic NOC-based MPSoC with Fault-Tolerance Support », *in : DAC Workshop on "Diagnostic Services in Network-on-Chips (DSNoC)"*, Anaheim, USA, june 2010.

[PPD10a] M. Pham, S. Pillement, D. Demigny, « Evaluation of Fault-Mitigation Schemes for Fault-Tolerant Dynamic MPSoC », *in : International Conference on Field Programmable Logic and Applications (FPL)*, 2010.

[PPD10b] M. Pham, S. Pillement, D. Demigny, « FT-DyMPSoC : Analytical Model for Fault-Tolerant Dynamic MPSoC », *in : IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Charlotte, North Carolina, may 2010.

[DPCD10b] L. Devaux, S. Pillement, D. Chillet, D. Demigny, « Mesh and Fat-Tree comparison for dynamically reconfigurable applications », *in : Workshop on Reconfigurable Communication-Centric SoCs (ReCoSoC)*, Karlsruhe, Germany, mai 2010.

[DPCD10c] L. Devaux, S. Pillement, D. Chillet, D. Demigny, « OS services for Reconfigurable System-on-Chip Communications », *in : Design of Circuits and Integrated Systems (DCIS)*, Lanzarote, Spain, mai 2010.

[ECPS10] A. Eiche, D. Chillet, S. Pillement, O. Sentieys, « Task placement for dynamic and partial reconfigurable architecture », *in : Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Edinburgh, Scotland, octobre 2010.

[DCPD09] L. Devaux, D. Chillet, S. Pillement, D. Demigny, « Flexible communication support for dynamically reconfigurable FPGAs », *in : Southern Programmable Logic Conference (SPL)*, p. 65–70, Sao Carlos, Brazil, 2009.

[DSP+09a] L. Devaux, S. B. Sassi, S. Pillement, D. Chillet, D. Demigny, « DRAFT : Flexible Interconnection Network for Dynamically Reconfigurable Architectures »,

in : IEEE International Conference on Field-Programmable Technology (FPT), Sydney, Australia, décembre 2009.

[LPS09b] J. Lallet, S. Pillement, O. Sentieys, « xMAML : a Modeling Language for Dynamically Reconfigurable Architectures », *in : Euromicro Conference on Digital System Design : Architectures, Methods and Tools (DSD)*, p. 680–687, Patras, Greece, septembre 2009.

[OVPN+09] Y. Oliva-Venegas, J.-C. Prevotet, F. Nouvel, S. Pillement, D. Chillet, « Exploration for Dynamic Reconfiguration Management », *in : Sophia Antipolis MicroElectronics Forum (SAME)*, September 2009.

[PPD09a] M. Pham, S. Pillement, D. Demigny, « A Fault-Tolerant Layer For Dynamically Reconfigurable Multi-Processor System-On-Chip », *in : International Conference on ReConFigurable Computing and FPGAs, (ReConFig)*, p. 284–289, Cancun, Mexico, décembre 2009.

[PPD09b] M. Pham, S. Pillement, D. Demigny, « Reconfigurable ECU Communications in Autosar Environment », *in : 9th International Conference on ITS Telecommunications (ITST)*, Lille, France, octobre 2009.

[PCOP09] S. Pillement, D. Chillet, Y. Oliva, J. C. Prevotet, « High-Level Exploration for Dynamic Reconfiguration Management », *in : Engineering of Reconfigurable Systems & Algorithms (ERSA)*, CSREA Press, p. 301–302, Las Vegas, Nevada, USA, juillet 2009.

[PC09] S. Pillement, D. Chillet, « High-level Model of Dynamically Reconfigurable Architectures », *in : Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Nice, France, septembre 2009.

[CPS08] D. Chillet, S. Pillement, O. Sentieys, « Reconfigurable Artificial Neural Network Model for Task Scheduling on Reconfigurable SoC », *in : Conference on Design and Architectures for Signal and Image Processing (DASIP)*, p. 92–99, Bruxelles, Belgium, novembre 2008.

[LPS08] J. Lallet, S. Pillement, O. Sentieys, « Efficient Dynamic Reconfiguration for Multi-context Embedded FPGA », *in : Symposium on Integrated circuits and system design (SBCCI)*, ACM, p. 210–215, Gramado, Brazil, 2008.

[PPS08] S. Pillement, J. Philippe, O. Sentieys, « A New Approach of Coding to Improve Speed and Noise Tolerance of On-Chip Busses », *in : International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, p. 1–6, Tozeur, Tunisia, mars 2008.

[PBG+08] J. Prevotet, A. Benkhelifa, B. Granado, E. Huck, B. Miramond, F. Verdier, D. Chillet, S. Pillement, « A Framework for the Exploration of RTOS Dedicated to the Management of Hardware Reconfigurable Resources », *in : International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, December 3-5 2008.

[BCPS07] I. Benkermi, D. Chillet, S. Pillement, O. Sentieys, « Hardware Task Scheduling for Heterogeneous SoC Architectures », *in : European Signal Processing Conference (EUSIPCO)*, p. 1653–1657, Poznan, Poland, septembre 2007.

[KHK+07] A. Kupriyanov, F. Hannig, D. Kissler, J. Teich, J. Lallet, O. Sentieys, S. Pillement, « Modeling of Interconnection Networks in Massively Parallel Processor Architectures », *in : International Conference on Architecture of Computing Systems (ARCS)*, *Lecture Notes in Computer Sciences*, *4415*, p. 268–282, Zurich, mars 2007.

[CPS07a] D. Chillet, S. Pillement, O. Sentieys, « A Neural Network Model for Real-Time Scheduling on Heteregeneous SoC Architectures », *in : International Joint Conference on Neural Networks (IJCNN)*, p. 102–107, Orlando, Florida, 2007.

[AKPD06] N. Abel, L. Kessal, S. Pillement, D. Demigny, « Clear stream towards dynamically reconfigurable systems on chip », *in : Workshop on Reconfigurable Communication-Centric SoCs (ReCoSoC)*, p. 98–104, France, 2006.

[PPS06] J. M. Philippe, S. Pillement, O. Sentieys, « Area Efficient Temporal Coding Schemes for Reducing Crosstalk Effects », *in : IEEE International Symposium on Quality Electronic Design (ISQED)*, p. 334–339, San José, USA, mars 2006.

[PKBPS06] J. M. Philippe, E. Kinvi-Boh, S. Pillement, O. Sentieys, « An Energy-Efficient Ternary Interconnection Link for Asynchronous Systems », *in : IEEE International Symposium on Circuits and Systems (ISCAS)*, p. 1014–1018, Kos, Greece, mai 2006.

[BBC+05a] I. Benkermi, A. Benkhelifa, D. Chillet, S. Pillement, J. Prévotet, F. Verdier, « System-Level Modelling for Reconfigurable SoCs », *in : Conference on Design of Circuits and Integrated Systems (DCIS)*, Lisboa - Portugal, novembre 2005.

[PPS05] J. M. Philippe, S. Pillement, O. Sentieys, « A Low-Power And High-Speed Quaternary Interconnection Link Using Efficient Converters », *in : IEEE International Symposium on Circuits and Systems (ISCAS)*, p. 4689–4692, Kobe, Japan, mai 2005.

[VPB+05] F. Verdier, J. Prévotet, A. Benkhelifa, D. Chillet, S. Pillement, « Exploring RTOS issues with a high-level model of a reconfigurable SoC platform », *in : Workshop on Reconfigurable Communication-Centric SoCs (ReCoSoC)*, p. 71–78, Montpellier, France, juin 2005.

[APSB04] F. B. Abdallah, S. Pillement, O. Sentieys, A. Bouallegue, « Acceleration of a VLIW Processor With Dynamic Reconfiguration », *in : IEEE International Conference on Microelectronics (ICM)*, p. 633–636, Tunis, Tunisie, décembre 2004.

[MGD+03] D. Menard, M. Guitton, S. Pillement, O. Sentieys, « Design and Implementation of WCDMA Platforms : Challenges and Trade-offs », *in : International Signal Processing Conference (ISPC)*, Dallas, Texas, USA, avril 2003.

[CPS02] D. Chillet, S. Pillement, O. Sentieys, « A Virtual Component for Motion Estimation Algorithm », *in : Engineering of Reconfigurable Systems & Algorithms (ERSA)*, Las Vegas, Nevada, USA, juin 2002.

[DCPS02b] R. David, D. Chillet, S. Pillement, O. Sentieys, « A Compilation Framework for a Dynamically Reconfigurable Architecture », *in : International Conference on Field Programmable Logic and Applications (FPL)*, *Lecture Notes in Computer Science*, *2438*, Springer Verlag, p. 1058–1067, La grande Motte, France, 2002.

[DCPS02c] R. David, D. Chillet, S. Pillement, O. Sentieys, « DART : a dynamically reconfigurable architecture dealing with future mobile telecommunications constraints », *in : IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, p. 0156–0164, Fort Lauderdale, Florida, USA, avril 2002.

[DCPS02d] R. David, D. Chillet, S. Pillement, O. Sentieys, « A High-Performance dynamically reconfigurable embedded architecture », *in : Sophia Antipolis Forum on Microelectronics (SAME)*, Nice, France, septembre 2002.

[DCPS02e] R. David, D. Chillet, S. Pillement, O. Sentieys, « Mapping Future Generation Mobile Telecommunication Applications on a Dynamically Reconfigurable Architecture », *in : 27th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Orlando, Florida, USA, mai 2002.

[SPC02] O. Sentieys, S. Pillement, D. Chillet, « Behavioral IP Specification and Integration Framework for High-Level Design Reuse », *in : IEEE International Symposium on Quality Electronic Design (ISQED)*, p. 388–393, San Jose, USA, 2002.

[DCPS01] R. David, D. Chillet, S. Pillement, O. Sentieys, « A Dynamically Reconfigurable Architecture for Low-Power Multimedia terminals », *in : International Conference on Very Large Scale Integration (VLSI-SOC)*, p. 51–62 Montpellier, France, 2001.

[PSC+01a] S. Pillement, O. Sentieys, D. Chillet, E. Casseau, P. Coussy, E.Martin, G. Savaton, S. Roux, « Design and Synthesis of Behavioral Level Virtual Components », *in : International Conference on Very Large Scale Integration (VLSI-SOC)*, p. 23–28, Montpellier, France, décembre 2001.

[PTRC99] S. Pillement, L. Torres, M. Robert, G. Cambon, « Fast Prototyping : A Case Study - The JPEG Compression Algorithm », *in : IEEE International Workshop on Rapid System Prototyping (RSP)*, p. 87–92, Clearwater, Floride, USA, 1999.

[RSC+99b] S. Raimbault, G. Sassatelli, G. Cambon, M. Robert, S. Pillement, L. Torres, « Embedded systems design and verification : Reuse oriented prototyping methodologies », *in : International Conference on Very Large Scale Integration (VLSI*, p. 407–414, Orlando, Floride, USA, décembre 1999.

[TPR+98] L. Torres, S. Pillement, S. Raimbault, L. Revelli, M. Robert, G. Cambon, « Fast Prototyping for Hardware / Software Codesign », *in : Sophia Antipolis Forum on Microelectronics (SAME)*, p. 65–69, Sophia Antipolis, France, octobre 1998.

[PTRC96a] S. Pillement, L. Torres, M. Robert, G. Cambon, « Concurrent design of hardware / software dedicated systems », *in : International Workshop on Field Programmable Logic and Applications (FPL)*, *Lecture Notes in Computer Science*, *1142*, Springer Verlag, p. 410–415, Darmstadt, Allemagne, septembre 1996.

[MCPS96] L. Maillet-Contoz, S. Pillement, J. Sallantin, « A unified workbench for designing hardware / software systems », *in : IFIP Working Conference on Logic and Architecture Synthesis (IWLAS*, p. 363–370, Grenoble, France, décembre 1996.

Conférences nationales avec actes

[DSP+09b] L. Devaux, S. B. Sassi, S. Pillement, D. Chillet, D. Demigny, « Réseau d'interconnexion flexible pour architecture reconfigurable dynamiquement et partiellement », *in : Symposium en Architecture de machines (SympA)*, Toulouse, France, septembre 2009.

[ECPS09a] A. Eiche, D. Chillet, S. Pillement, O. Sentieys, « Flot d'ordonnancement pour architecture reconfigurable », *in : Symposium en Architecture de machines (SympA)*, Toulouse, France, septembre 2009.

[LPS09b] J. Lallet, S. Pillement, O. Sentieys, « Plate-forme de Conception d'Architectures Reconfigurables Dynamiquement pour le Domaine du TSI », *in : Colloque sur le Traitement du Signal et des Images (GRETSI)*, p. 210–215, Dijon, France, septembre 2009.

[CPS07b] D. Chillet, S. Pillement, O. Sentieys, « Vers une implémentation matérielle d'un réseau de neurones pour le service d'ordonnancement des tâches au sein d'un SoC », *in : Colloque sur le Traitement du Signal et des Images (GRETSI)*, p. 353–356, Troyes, France, septembre 2007.

[BBC+05b] I. Benkermi, A. Benkhelifa, D. Chillet, S. Pillement, J. Prevotet, F. Verdier, « Modélisation niveau système de SoC reconfigurables », *in : Symposium en Architecture de machines (SympA)*, p. 107–118, Le Croisic, Presqu'île de Guérande, France, avril 2005.

[BPS03] I. Benkermi, S. Pillement, O. Sentieys, « Application des réseaux de neurones à l'ordonnancement de tâches temps réel sur une architecture multiprocesseurs hétérogènes », *in : Symposium en Architecture de machines (SympA)*, p. 372–379, la Colle sur Loup, France, octobre 2003.

[PDS03] S. Pillement, R. David, O. Sentieys, « Papier invité : Architectures reconfigurables : opportunités pour la faible consommation », *in : Colloque Faible Tension Faible Consommation (FTFC)*, p. 31–39, France, mai 2003.

[MGD+03] D. Menard, M. Guitton, R. David, S. Pillement, O. Sentieys, « Évaluation comparative de plates-formes reconfigurables et programmables pour les télécommunications de 3ème génération », *in : Colloque sur le Traitement du Signal et des Images (GRETSI)*, France, avril 2003.

[CPSa02] D. Chillet, S. Pillement, O. Sentieys, al., « Vers une approche unifiée pour la conception globale des terminaux de télécommunications », *in : Journées Francophones sur l'Adéquation Algorithme Architecture*, Gabes, Tunisie, 2002.

[DPSC01] R. David, S. Pillement, O. Sentieys, D. Chillet, « Architectures Enfouies Reconfigurables Dynamiquement », *in : Symposium en Architecture de machines (SympA)*, p. 23–32, Paris, France, avril 2001.

[PSC01a] S. Pillement, O. Sentieys, D. Chillet, « Vers la définition de composants virtuels au niveau algorithmique », *in : Colloque sur le Traitement du Signal et des Images (GRETSI)*, Toulouse, France, septembre 2001.

Colloque et séminaires

[ECPS09b] A. Eiche, D. Chillet, S. Pillement, O. Sentieys, « Flot d'Ordonnancement Temps Réel d'un Ensemble de Tâches Matérielles pour Architecture Reconfigurable », Workshop GDR SoCSiP, June 2009.

[PMS07] S. Pillement, D. Menard, O. Sentieys, « TDSI et reconfigurables : Etude d'implémentations », Journée du GDR ISIS, mars 2007.

[PLS06] S. Pillement, J. Lallet, O. Sentieys, « Vers un language de description d'architectures reconfigurables », Séminaire du GDR SoC-SiP, may 2006.

[Pil04] S. Pillement, « EPML POMARD Thème 2 : Architectures », Séminaire du RTP SoC, mai 2004.

[DCPS02f] R. David, D. Chillet, S. Pillement, O. Sentieys, « Flot de Conception pour Plateforme Reconfigurable », Acte du colloque de CAO de circuits intégrés et systèmes, avril 2002.

[TPRC98] L. Torres, S. Pillement, M. Robert, G. Cambon, « Prototyping Workbench for Hardware / Software Codesign », Jornadas de Actualizacion en Microelectronica,(Papier invité), janvier 1998.

[Pil97a] S. Pillement, « Evaluation des systèmes numériques complexes : du prototype à l'ASIC », Journées des doctorants de Montpellier, novembre 1997.

[Pil97b] S. Pillement, L. Torres, M. Robert, G. Cambon, « Expérimentation d'un environnement de prototypage de systèmes mixtes », Acte du colloque de CAO de circuits intégrés et systèmes, janvier 1997.

[PTRC96b] S. Pillement, L. Torres, M. Robert, G. Cambon, « Aide à la validation de systèmes logiciels/ matériels : système de prototypage LIRMM », Actes des séminaires action scientifique, France Télécom, novembre 1996.

Bibliographie générale

[1] B. Ahmad, A. Erdogan, and S. Khawam. Architecture of a Dynamically Reconfigurable NoC for Adaptive Reconfigurable MPSoC. In *First NASA/ESA Conference on Adaptive Hardware and Systems*, 2006.

[2] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, and J. Teich. A Practical Approach for Circuit Routing on Dynamic Reconfigurable Devices. *Workshop on Rapid System Prototyping.*, pages 84–90, 2005.

[3] H. Amano. A Survey on Dynamically Reconfigurable Processors. *IEICE transactions on Communications*, E89-B(12) :3179 – 3187, December 2006.

[4] AMIGOS. `https ://info.enstb.org/projets/AMIGOS`.

[5] D. Andrews, R. Sass, E. Anderson, J. Agron, W. Peck, J. Stevens, F. Baijot, and E Komp. Achieving Programming Model Abstractions for Reconfigurable Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16 :34–44, 2008.

[6] ATMEL. *AT40K05/10/20/40AL. 5K - 50K Gate FPGA with DSP Optimized Core Cell and Distributed FreeRam, Enhanced Performance Improvement and Bi-directional I/Os (3.3 V).*, 2006. revision F.

[7] M. Bashiri, S.G. Miremadi, and M. Fazeli. A Checkpointing Technique for Rollback Error Recovery in Embedded Systems. In *International Conference on Microelectronics*, pages 174–177, 2006.

[8] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka. Dynamic and Partial FPGA Exploitation. *Proceedings of the IEEE*, 95 :438–452, 2007.

[9] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, MS Reorda, M. Violante, and P. Zambolin. Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA. In *Design, Automation and*

Test in Europe Conference and Exhibition, 2004. Proceedings, volume 1. IEEE Computer Society Washington, DC, USA, 2004.

[10] L. Benini and G. De Michelli. Networks on Chips : A new SoC Paradigm. *IEEE Computer*, 35(1) :70–78, January 2002.

[11] I. Benkermi. *Système d'exploitation temps réel pour architectures reconfigurables dynamiquement á faible consommation.* PhD thesis, Université de Rennes 1, July 2007.

[12] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, KA LaBel, M. Friendlich, H. Kim, and A. Phan. Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA : Design, Test, and Analysis. *IEEE Transactions on Nuclear Science*, 55(4 Part 1) :2259–2266, 2008.

[13] J. M. Berger. A note on error detection codes for asymmetric binary channels. In *Inform. Contr.*, volume 4, pages 68–73, 1961.

[14] P. Bernardi, M.S. Reorda, L. Sterpone, M. Violante, and I. Torino. On the evaluation of SEU sensitiveness in SRAM-based FPGAs. In *On-Line Testing Symposium*, pages 115–120. IEEE Computer Society Washington, DC, USA, 2004.

[15] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen. Dynoc : A Dynamic Infrastructure For Communication In Dynamically Reconfigurable Devices. In *Field Programmable Logic and Applications*, LNCS, pages 153–158, 2005pionteck.

[16] G. Brebner. An interactive datasheet for the xilinx XC6200. In *International Conference on Field Programmable Logic and Applications*, volume 1482 of *Lecture Notes in Computer Science*, pages 401–405. Lecture Notes in Computer Science, 1998.

[17] I. Brynjolfson and Z. Zilic. FPGA clock management for low power applications. In *International Symposium on Field programmable gate arrays*, page 219, 2000.

[18] H. Castro, A.A. Coelho, and R.J. Silveira. Fault-tolerance in FPGA's through CRC voting. In *Proceedings of the twenty-first annual symposium on Integrated circuits and system design*, pages 188–192. ACM New York, NY, USA, 2008.

[19] F. Charot and V. Messe. A Flexible Code Generation Framework for the Design of Application Specific Programmable Processors. In *International workshop on Hardware/Software Codesign*, pages 27–32, Rome, Italy, May 1999.

[20] K. Compton and S. Hauck. Reconfigurable Computing : A Survey of Systems and Software. *ACM Computing Surveys*, 34(2) :171–210, June 2002.

[21] S. Corbetta, V. Rana, M.D. Santambrogio, and D. Sciuto. A light-weight Network-on-Chip architecture for dynamically reconfigurable systems. In *International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation, 2008. SAMOS 2008*, pages 49–56, 2008.

[22] A. Courtay, O. Sentieys, J. Laurent, and N. Julien. High-level interconnect delay and power estimation. *Journal of Low Power Electronic*, 4 :21–33, 2008.

[23] D. Cozzi, C. Farè, A. Meroni, V. Rana, M.D. Santambrogio, and D. Sciuto. Reconfigurable NoC design flow for multiple applications run-time mapping on FPGA devices. In *ACM Great Lakes symposium on VLSI*, pages 421–424, 2009.

[24] J. Cui, Q. Deng, X. He, and Z. Gu ;. An Efficient Algorithm for Online Management of 2D Area of Partially Reconfigurable FPGAs. In *Design, Automation & Test in Europe*, 2007.

[25] W. Dally and J. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[26] R. David. *Architecture reconfigurable dynamiquement pour applications mobiles*. PhD thesis, Université de Rennes 1, July 2003.

[27] A. Devgan. Efficient Coupled Noise Estimation for On-Chip Interconnects. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'97)*, pages 147–153, San Jose, California, USA, November 1997.

[28] J.P. Diguet, Y. Eustache, and M. El Khodary. Feedback Control Learning Model for QoS, Power & Performance Management of Reconfigurable Embedded Systems. In *International Symposium on DSP and Communication Systems, DSPCS'2005*, Noosa Heads, Australia, December 2005.

[29] N. Dutt and P. Mishra. Architecture Description Languages for Programmable Embedded Systems. *IEE Proc. : Computers and Digital Techniques*, pages 285–297, 2005.

[30] M. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Publisher, Elsevier Science, 2004.

[31] D. Flandre, S. Adriaensen, A. Afzalian, J. Laconte, D. Levacq, C. Renaux, L. Vancaillie, J.-P. Raskin, L Demeus, P. Delatte, V. Dessard, and G. Picun. Intelligent SOI CMOS Integrated Circuits and Sensors of Heterogeneous Environments and Applications. In *1st IEEE International Conference on Sensors*, volume 2, pages 1407–1412, USA, 2002.

[32] A. Fraboulet. *Optimisation de la mémoire et de la consommation des systèmes multimédia embarqués*. PhD thesis, INSA de Lyon, November 2001.

[33] H. Freitas and P. Navaux. Evaluating On-Chip Interconnection Architectures for Parallel Processing. In *Conference on Computational Science and Engineering*, 2008.

[34] J. Fridman. Sub-Word Parallelism in Digital Signal Processing. *IEEE Signal Processing Magazine*, 17(2) :27–35, March 2000.

[35] A.K. Goel. *High-speed VLSI interconnections*. Wiley-IEEE Press, 2007.

[36] A. Greenfield. *Everyware : The dawning age of ubiquitous computing*. Peachpit Pr, 2006.

[37] Y. Guillemenet, S. Ahmed, L. Torres, A. Martheley, J. Eydoux, J.B. Cuelle, L. Rougé, and G. Sassatelli. MRAM Based eFPGAs : Programming and Silicon Flows, Exploration Environments, MRAM Current State in Industry and Its Unique Potentials for FPGAs. In *Conference on Reconfigurable Computing and FPGAs*, 2009.

[38] J. L. Hennessy and D. A. Patterson. *Computer Architecture : A Quantitative Approach*, chapter Appendix E : Interconnection Networks. Morgan Kaufmann, 2006.

[39] T.-C. Huang. A low-power dependable Berger code for fully asymmetric communication. *IEEE Communications Letters*, 12 :773–775, 2008.

[40] Y. Ichinomiya, S. Tanoue, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi. Improving the Robustness of a Softcore Processor against SEUs by Using TMR and Partial Reconfiguration. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 47–54, 2010.

[41] S. Jovanovic, C. Tanougast, C. Bobda, and S. Weber. CuNoC : A dynamic scalable communication structure for dynamically reconfigurable FPGAs. *Microprocessors and Microsystems*, 33(1) :24 – 36, 2009.

[42] A. Kanamaru, H. Kawai, Y. Yamaguchi, and M. Yasunaga. Tile-Based Fault Tolerant Approach Using Partial Reconfiguration. In *Proc. Int. Workshop on Reconfigurable Computing : Architectures, Tools and Applications. LNCS*, volume 5453, pages 293–299, 2009.

[43] Heikki Kariniemi. *On-Line Reconfigurable Extended Generalized Fat-Tree Network-on-Chip for MultiProcessor System-on-Chip Circuits.* PhD thesis, Tampere University of Technology, 2006.

[44] E. Kinvih-Boh, M. Aline, O. Sentieys, and E. D. Olson. MVL circuit design and characterization at the transistor level using SUS-LOC. In *Proceedings of the IEEE International Symposium on Multiple-Valued Logic*, pages 105–110, Tokyo, Japan, May 2003.

[45] Dirk Koch, Christian Beckhoff, and Juergen Teich. ReCoBus-Builder- a Novel Tool and Technique to Build Statically and Dynamically Reconfigurable Systems for FPGAs. In *The international conference on Field-Programmable Logic and its Applications*, 2008.

[46] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. In *International Symposium on Field Programmable Gate Arrays*, pages 21–30, New York, USA, 2006.

[47] A. Kupriyanov. MAML - An Architecture Description Language for Modeling and Simulation of Processor Array Architectures. Technical report, Department of Computer Science 12, Hardware-Software Co-Design, University of Erlangen-Nüremberg, 2006.

[48] L. Lagadec. *Abstraction, modélisation et outils de CAO pour les circuits intégrés reconfigurables.* PhD thesis, Universit/'e de Rennes 1, 2000.

[49] L. Lagadec, D. Picard, and P.Y. Lucas. Teaching reconfigurable computer : the Biniou approach. In *international workshop on Reconfigurable Communication-centric Systems on Chip - ReCoSoc*, 2010.

[50] J. Lallet. *Plate-forme générique de modélisation et de conception d'architectures reconfigurables dynamiquement.* PhD thesis, Université de Rennes 1, July 2008.

[51] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, and al. A telecom baseband circuit based on an asynchronous network-on-chip. In *IEEE International Solid-State Circuits Conference*, pages 258–601, 2007.

[52] L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. A Crosstalk Aware Interconnect with Variable Cycle Transmission. In *EEE/ACM International Conference on Design, Automation and Test in Europe*, Paris, France, 2004.

[53] S.-W Liao. *SUIF Explorer : an Interactive and Interprocedural Parallelizer*. PhD thesis, Computer Systems Laboratory, Stanford University, May 2000.

[54] E. Lubbers and M. Planner. ReconOS : An RTOS Supporting Hard-and Software Threads. In *International Conference on Field Programmable Logic and Applications*, pages 441–446, 2007.

[55] Dylan McGrath. Altera to offer partial reconfiguration at 28-nm. *EETimes (* `http ://www.eetimes.com/showArticle.jhtml ?articleID=222600544`*)*, 2010.

[56] H. Meyer. *Analysis and Design of Low Power Digital Multipliers*. PhD thesis, Carnegie Mellon University, Pennsylvania, USA, August 1999.

[57] F. Moraes. Atlas - An Environment for NoC Generation and Evaluation. http ://ww1.inf.pucrs.br/ moraes/.

[58] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost. HERMES : an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, 38 :69–93, 2004.

[59] Vijaykrishnan Narayanan and Yuan Xie. Reliability Concerns in Embedded System Designs. *Computer*, 39(1) :118–120, 2006.

[60] J. Nurmi, J. Isoaho, A. Jantsch, and H. Tenhunen. *Interconnect-centric design for advanced SoC and NoC*. Kluwer Academic Publishers, 2004.

[61] NVidia. The Next Generation CUDA Architecture, Code Named Fermi The Soul of a Supercomputer in the Body of a GPU. `http ://www.nvidia.com/object/fermi_architecture.html`, 2010.

[62] OAR Corporation. RTEMS Operating System. `http ://www.rtems.com/`, 2010.

[63] T. Ojanpera and R. Prasad. *WCDMA : Towards IP Mobility and Mobile Internet*. Artech House Publishers, 2001.

[64] E. D. Olson. *Multiple-valued logic circuit architecture ; supplementary symmetrical logic circuit structure (SUS-LOC)*. USPTO Patent 6,133,754, 2000.

[65] OverSoC. DOGME "Distributed Operating system Graphical Modelling Environment". http ://oversoc.ensea.fr/oversoc-graphical-modeling-environment-1, 2010.

[66] E. Ozer, R. Sendag, and D. Gregg. Multiple-valued logic buses for reducing bus energy in low-power systems. In *IEE Computers & Digital Techniques*, pages 270–282, 2006.

[67] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures. *IEEE Transactions on Computers*, 54 :1025–1040, 2005.

[68] M. Pedram and J. M. Rabaey. *Power Aware Design Methodologies*, chapter 8, pages 201–239. Kluwer Academic Publishers, June 2002.

[69] G. Perrouin, F. Chauvel, J. DeAntoni, and J.M. Jézéquel. Modeling the Variability Space of Self-Adaptive Applications. In *Dynamic Software Product Lines Workshop*, pages 15–22, 2008.

[70] J.M. Philippe. *Intégration des réseaux sur silicium : optimisation des performances des couches physique et liaison.* PhD thesis, Université de Rennes 1, July 2005.

[71] C. Pilotto, J.R. Azambuja, and F.L. Kastensmidt. Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications. In *Proceedings of the twenty-first annual symposium on Integrated circuits and system design*, pages 199–204, 2008.

[72] T. Pionteck, R. Koch, and C. Albrecht. Applying Partial Reconfiguration to Networks-on-Chips. In *The international conference on Field-Programmable Logic and its Applications*, pages 1–6, 2006.

[73] J. Pistorius, M. Hutton, A. Mishchenko, and R.Brayton. Benchmarking Method and Designs Targeting Logic Synthesis for FPGAs. In *Proc. of the International Workshop on Logic and Synthesis*, 2007.

[74] D.K. Pradhan and N.H Vaidya. Roll-Forward Checkpointing Scheme : A Novel Fault-Tolerant Architecture. *IEEE Transactions on Computers*, 43 :1163–1174, 1994.

[75] Y. Qu, J-P. Soininen, and J. Nurmi. Static Scheduling Techniques for Dependent Tasks on Dynamically Reconfigurable Devices. *Journal of Systems Architecture*, 2007.

[76] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits : A Design Perspective.* Prentice Hall, 2002.

[77] E. Salminen, A. Kulmala, and T. D. Hamalainen. Survey of Network-on-chip Proposals. *White Paper, OCP-IP*, `http ://www.ocpip.org/socket/whitepapers`, 2008.

[78] H. Schmeck. Organic computing - a new vision for distributed embedded systems. In *International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 201–203, 2005.

[79] O. Sentieys, J.P. Diguet, and J.L. Philippe. GAUT : A High Level Synthesis Tool Dedicated To Real Time Signal Processing Applications. In *European Design Automation Conference*, September 2000.

[80] N. R. Shanbhag. Reliable and Efficient System-on-Chip Design. *IEEE Computer*, 37(3) :42–50, March 2004.

[81] N. R. Shanbhag, D. Nagchoudhuri, R. E. Siferd, and G. S. Visweswaran. Quaternary Logic Circuits in 2μm CMOS Technology. *IEEE Journal of Solid-States Circuits*, 25(3) :790–799, June 1990.

[82] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang. Communication-aware Heuristics for Run-time Task Mapping on NoC-based MPSoC Platforms. *Journal of Systems Architecture*, In Press, Accepted Manuscript :–, 2010.

[83] M. R. Stan and W. P. Burleson. Bus-invert coding for low-power I/O. *IEEE Transactions on VLSI systems*, 3 :49–58, 1995.

[84] C. Steiger, H. Walder, and M. Platzner. Operating Systems for Reconfigurable Embedded Platforms : Online Scheduling of Real-Time Tasks. *IEEE Trans. on Computers*, 53(11) :1393–1407, 2004.

[85] I. M. Thoidis, D. Soudris, I. Karafyllidis, and A. Thanailakis. The Design of Low-Power Multiple-Valued Logic Encoder and Decoder Circuits. *International Conference on Electronics, Circuits and Systems*, 3 :1623–1626, September 1999.

[86] J. Villarreal, D. Suresh, G. Stitt, F. Vahid, and W. Najjar. Improving Software performance with Configurable Logic. *Design Automation for Embedded Systems*, 7(4) :325–339, November 2002.

[87] Xilinx. *Virtex FPGA Series Configuration and Readback, Application Note XAPP138*, 2005.

[88] Xilinx. *Difference-Based Partial Reconfiguration, Application Note XAPP290*, 2007.

[89] Xilinx. PPC405 lockstep system on ml310. *Application Note XAPP564*, 2007.

[90] Xilinx. *XST User Guide for Virtex-6 and Spartan-6 Devices*, 2009.

[91] Xilinx, Inc. Virtex-5 FPGA Configuration User Guide (UG191 v3.6), 2009.

[92] H. Zhang, V. George, and J. M. Rabaey. Low-Swing On-Chip Signaling Techniques : Effectiveness and Robustness. *IEEE Transactions on Very Large Scale Integration Systems*, 8(3) :264–272, June 2000.

Résumé

Les travaux de recherche, dont la synthèse est présentée dans ce manuscrit, portent sur la conception de systèmes reconfigurables dynamiquement. L'évolution constante des applications et le besoin toujours croissant de performances imposent le développement de nouvelles architectures performantes et flexibles. Ces contraintes ont amené à une complexification des architectures, de leurs mécanismes de reconfiguration et de leur gestion. Dans le premier axe de travail, nous proposons des architectures offrant un bon compromis performances, consommation d'énergie, flexibilité. Afin de simplifier la conception de ces architectures et de leur gestion nous avons proposé un langage de description haut niveau qui permet de générer l'architecture mais aussi de paramétrer ses outils de développement.

Les systèmes sur puce moderne incluent un grand nombre de fonctionnalités hétérogènes, et doivent faire face à des problèmes liés à la réduction des dimensions technologiques. Pour répondre à ces difficultés, le concept de réseau intégré sur silicium semble prometteur. Dans notre deuxième axe, nous étudions de nouveaux codages et l'utilisation de nouvelles technologies pour réduire la consommation des interconnexions tout en améliorant leur fiabilité. Nous travaillons également à la définition de réseaux flexibles adaptés au paradigme de la reconfiguration dynamique.

L'émergence de systèmes intégrant une zone reconfigurable dynamiquement nécessite l'emploi d'outils et de mécanismes spécifiques. En particulier, la présence d'un système d'exploitation adapté devient nécessaire. Celui-ci doit être capable, au minimum, d'ordonnancer les tâches à exécuter, d'assurer le partage des moyens de communication et d'offrir un modèle de déploiement d'applications indépendant de l'architecture cible. Le deuxième enjeu de cet axe vient de la nécessité de développer des architectures tolérantes aux fautes. Ainsi la mise en place de gestions spécifiques permet de développer des systèmes reconfigurables dynamiquement sûrs de fonctionnement.

Abstract

The research presented in this manuscript focus on the design of dynamically reconfigurable systems. Constant evolution of applications and the ever increasing need for performance require the development of new efficient and flexible architectures. These constraints have led to more complex architectures, their reconfiguration mechanisms and management. In the first part of our work, we propose architectures providing a good compromise between performance, power consumption, and flexibility. To simplify the design of these architectures and their management, we have proposed a high level description language that allows to generate the architecture but also set up its development flow.

The modern SoCs include a large number of heterogeneous features, and face problems due to technology shrink. To meet these challenges, the concept of integrated network on silicon seems promising. In our second line, we are studying new coding and new technologies to reduce consumption of interconnects while improving their reliability. We are also working to define flexible networks adapted to the dynamic reconfiguration paradigm.

The emergence of reconfigurable systems requires the use of specific tools and mechanisms. In particular, the presence of a dedicated operating system becomes necessary. It would provide services such as tasks scheduling, communications management and provide a model independent of the target architecture for applications deployment. The second issue of this axis is the need to develop fault-tolerant architectures. Thus the establishment of specific management can develop reliable dynamically reconfigurable systems.